NASP-9218-07
NAS EnrouteStage A
Contract FA65WA-1395

9 DEC 1974

9020 DOCUMENT CONTROL SECTION

COPY No. 11

NAS OPERATIONAL SUPPORT SYSTEM

USER'S MANUAL

Library Subroutines
(LIBARY)

Model A3d2.1

15 August 1974

This manual describes the IBM 9020 Programming System Library Subroutines. The library routines offer a number of options: in mathematical calculations; in data movement, comparison, and conversion; and in input/output operations.

These change pages update this document to make it compatible with the NOSS tapes which support the NAS Model A3d2.1 System.

NAS Programming
IBM Federal Systems Division
NAFEC, Atlantic City, New Jersey

This (07 level) publication of the LIBARY User's Manual is current with the tape labeled L3D21x.

Level	Date	Change Proposal Number	Comment
01	August 1967		Original Publication
02	15 January 1969		First Update
03	26 June 1972		Second Update
04	15 April 1973		A3d1.2 Update
05	15 August 1973		A3d2.0 Update
06	24 May 1974		A3d2.1 Update
07	15 August 1974		A3d2.1 Update

x = latest tape level

This document, prepared by the International Business Machines Corporation, is submitted to the Federal Aviation Administration in accordance with the requirements of Contract FA65WA–1395.

These change pages update the NAS Operational Support System (NOSS) User's Manual for Library Subroutines (LIBARY), dated 24 May 1974, to make it compatible with the NOSS tapes which support the NAS Model A3d2.1 System.

This manual contains information needed by the programmer to use the library routines supplied as part of the IBM 9020 Programming System. The purpose of each routine is described and its calls and parameters are defined.

The prerequisites for using this manual are a general knowledge of the IBM 9020 Programming System and a familiarity with the JOVIAL language. This information is in the following publications:

IBM 9020 Data Processing System – System Reference Manual,
Form C28–6549

NAS Operational Support System User's Manual for IBM 9020 JOVIAL Language (JOVIAL),
NASP–9217–latest revision, Contract FA65WA–1395, IBM Corporation, NAFEC, Atlantic City, N.J.

Other publications that relate to the material in this manual are:

NAS Operational Support System User's Manual for the System Edit Program (SYSEDT),
NASP–9222–latest revision, Contract FA65WA–1395, IBM Corporation, NAFEC, Atlantic City, N.J.

NAS Operational Support System User's Manual for the Loader Program (LOADER),
NASP–9219–latest revision, Contract FA65WA–1395, IBM Corporation, NAFEC, Atlantic City, N.J.

NAS Operational Support System User's Manual for the IBM 9020 Data Processing System Debugging System (DEBUGG), NASP–9216–latest revision, Contract FA65WA–1395, IBM Corporation, NAFEC, Atlantic City, N.J.

NAS Operational Support System User's Manual for the Basic Assembly Language (BALASM),
NASP–9214–latest revision, Contract FA65WA–1395, IBM Corporation, NAFEC, Atlantic City, N.J.

NAS Operational Support System User's Manual for the SPT Edit Program (SPTEDT),
NASP–9221–latest revision, Contract FA65WA–1395, IBM Corporation, NAFEC, Atlantic City, N.J.

NAS Operational Support System User's Manual for the Compool Edit Program (CMPEDT),
NASP–9215–latest revision, Contract FA65WA–1395, IBM Corporation, NAFEC, Atlantic City, N.J.

NAS Operational Support System User's Manual for the Utility NOSS Monitor (UTILITY),
NASP–9229–latest revision, Contract FA65WA–1395, IBM Corporation, NAFEC, Atlantic City, N.J.

Monitor Handbook, NASP–5201–latest revision, Contract FA65WA–1395, IBM Corporation, NAFEC, Atlantic City, N.J.

Section		Page
	PREFACE	ii
	CONTENTS	iii
1.0	INTRODUCTION	1–1
2.0	9020 LIBRARY	2–1
2.1	Programmer-Called Subroutines	2–1
2.1.1	Subroutine Abstracts by Type	2–2
2.1.2	Routine/Service Characteristics	2–7
2.1.3	BAL Programming Considerations in Using Library Facilities	2–10
2.1.4	Subroutine Descriptions	2–11
2.1.4.1	ADR Function	2–11
2.1.4.2	ARCCOS Function	2–12
2.1.4.3	ARCSIN Function	2 12
2.1.4.4	ARC Tangent (ARCTAN) Function	2–13
2.1.4.5	Arc Tangent in Degrees (ARCTND) Function	2–14
2.1.4.6	Arctangent in ACPs (ARCTNP, ATANFP)	2 14
2.1.4.7	ASCIN Procedure	2 15
2.1.4.8	ASCOUT Procedure	2–16
2.1.4.9	Backspace Block (BSB) Procedure	2–16
2.1.4.10	BSF Procedure	2–17
2.1.4.11	BSR Procedure	2–17
2.1.4.12	Compare Logical Character (CLC) Function	2–17
2.1.4.13	Compare Logical Character Immediate (CLCI) Function	2–18
2.1.4.14	Core Dump in Hexadecimal (CORE) Procedure	2 19
2.1.4.15	Cosine (COS) Function	2 20
2.1.4.16	Compute SIN/COS from ACPs (CSINE) Procedure	2 21

CONTENTS (Continued)

Section		Page
2.1.4.17	EXPON Function	2–21
2.1.4.18	Convert EBCDIC to Internal Floating Point (FLTFL) Procedure	2–22
2.1.4.19	Floating Field (FLTFLD) Procedure	2–23
2.1.4.20	Convert an EBCDIC Decimal Number to Floating Point (FLTINJ) Procedure	2–25
2.1.4.21	FLTOTJ Procedure	2–26
2.1.4.22	Fortran Square Root (FSQRT) Function	2–26
2.1.4.23	Get LISTIO Address (GETLIO) Function	2–27
2.1.4.24	Hexadecimal Input Conversion (HEXIN) Procedure	2–27
2.1.4.25	Hexadecimal Output Conversion (HEXOUT) Procedure	2–28
2.1.4.26	Process Floating Fields (HOLFL) Procedure	2–28
2.1.4.27	HOLINJ Procedure	2–29
2.1.4.28	HOLOTJ Procedure	2–29
2.1.4.29	Convert EBCDIC to Internal Binary (INTFL) Procedure	2–30
2.1.4.30	Convert EBCDIC integer to a Signed Binary Number (INTINJ) Procedure	2–30
2.1.4.31	Convert Integer Item to EBCDIC with Leading Blanks (INTOTJ) Procedure	2–31
2.1.4.32	Convert Integer Item to EBCDIC with Leading Zeros (INTOTZ) Procedure	2–31
2.1.4.33	Left-Justify Hollerith Data (LEFTJ) Function	2–32
2.1.4.34	LOC Function	2–33
2.1.4.35	Locate Table Address in Words (LOCW) Function	2–33
2.1.4.36	LOGN Function	2–33
2.1.4.37	Loop Control (LTIME) Procedure	2–34
2.1.4.38	Compute SIN/COS From ACPs (MCSINE) Procedure	2–35
2.1.4.39	Convert Mixed Number to EBCDIC (MIXOTJ) Procedure	2–35
2.1.4.40	MODULO Function	2–36

Section		Page
2.1.4.41	Move Character (MVC) Procedure	2–37
2.1.4.42	Move Item to Table (MVI) Procedure	2–38
2.1.4.43	Move Unlimited Character (MVM) Procedure	2–39
2.1.4.44	Move Table to Item (MVT) Function	2–40
2.1.4.45	AND Character (NC) Procedure	2–40
2.1.4.46	AND Item to Table (NI) Procedure	2–41
2.1.4.47	OR Character (OC) Procedure	2–41
2.1.4.48	OR Item to Table (OI) Procedure	2–41
2.1.4.49	Operational SVC Simulation (OPSVC) Procedure	2–42
2.1.4.50	Punch Object Card (POBJ) Procedure	2–44
2.1.4.51	General Print (PRINT) Procedure	2–46
2.1.4.52	Punch Text without Zero Cards (PTXNZ) Procedure	2–47
2.1.4.53	Punch Text (PTXT) Procedure	2–49
2.1.4.54	General Punch (PUNCH) Procedure	2–50
2.1.4.55	RANDOM Function	2–51
2.1.4.56	Random Integer (RANINT) Function	2–52
2.1.4.57	Read General Input (REED) Procedure	2–53
2.1.4.58	Read Tape to Buffer (REEDR) Procedure	2–54
2.1.4.59	REW Procedure	2–55
2.1.4.60	Read Console Typewriter (RTYPE) Procedure	2–55
2.1.4.61	RUN Procedure	2–56
2.1.4.62	Write General Output (RYTE) Procedure	2–56
2.1.4.63	NOSS Set I/O Return (SETRET) Procedure	2–58
2.1.4.64	Sine (SIN) Function	2–59
2.1.4.65	Super Print (SPRINT) Procedure	2–59

Section		Page
2.1.4.66	Square Root (SQRT) Function	2–64
2.1.4.67	STF Procedure	2–64
2.1.4.68	STR Procedure	2–64
2.1.4.69	System Communication (SYSCOM) Procedure	2–65
2.1.4.70	Set Up I/O Record Format (SYSDTF) Procedure	2–67
2.1.4.71	TIMDIF Function	2–68
2.1.4.72	Execute TRT plus Move and TR (TMTR) Procedure	2–68
2.1.4.73	Translate (TR) Procedure	2–69
2.1.4.74	Translate and Test (TRT) Procedure	2–70
2.1.4.75	Execute Test-and-Set (TSET) Function	2–70
2.1.4.76	Execute TRT and TR (TSTR) Procedure	2–70
2.1.4.77	TYPE Procedure	2–71
2.1.4.78	TYPEW Procedure	2–71
2.1.4.79	Read Variable Blocked Input Tape and Move Logical Records to User Area (VREED) Routine	2–72
2.1.4.80	Read Variable Blocked Input Tape and Return Address of Logical Record in Buffer (VREEDR) Routine	2–73
2.1.4.81	WEF Procedure	2–75
2.1.4.82	WRE Procedure	2–75
2.1.4.83	WRU Procedure	2–75
2.1.4.84	Exclusive OR Character (XC) Procedure	2–76
2.1.4.85	Exclusive OR Item to Table (XI) Procedure	2–76
2.2	Compiler-Called Routines	2–76
2.2.1	Floating-Point Base to Floating-Point Exponent (EXPFF) Routine	2–76
2.2.2	Floating-Point Base to Integral Exponent (EXPFI) Routine	2–77
2.2.3	Floating to Fixed-Conversion (FIX) Routine	2–78

2.2.4	Fixed A-Type to Floating Conversion (FLTA) Routine	2–78
2.2.5	Fixed Integer to Floating Conversion (FLTI) Routine	2–79
3.0	9020 LIBRARY TAPE	3–1
Appendix A	OBSOLETE ROUTINES	A–1

ILLUSTRATIONS

Figure		Page
1–1	Programming System For IBM 9020 Data Processing System	1–1

TABLES

Table		Page
2–1	Routine/Service Characteristics	2–8

The library subroutines described in this manual are the functions and procedures furnished as part of the IBM 9020 Programming System. They are recorded on the library tape and can be requested by any JOVIAL program. The routines perform a variety of tasks, including mathematical calculations, input/output positioning and control, data comparison, data movement, and data conversion. Programs written in Basic Assembly Language (BAL) may also request the mathematical functions by means of BAL calling sequences.

The JOVIAL programmer can request any library routine by including its specific calling statement in his program. However, a few routines (such as those for internal format conversion) are requested by the compiler only. When a JOVIAL source program calls a routine, the call and the source program are converted into BAL during compilation. Both program and call are then translated into object code by the BAL Assembler. At load time, the loader extracts the routine from the library tape and places it in storage with the requesting program.

The Library Edit Program (LIBEDT) is used to create a new library tape, or to add routines to (or delete routines from) the existing library tape. The Library Edit Program is called when the monitor (Figure 1–1) encounters a $LIBEDT control card. Routines that are to be included on the library tape must be in object code.

The library tape is used by the compiler and the loader. The compiler uses tabulated information at the beginning of the library tape to determine the routines needed and their parameter requirements. After reading all user-supplied object decks into storage, the loader reads all referenced routines from the library tape.

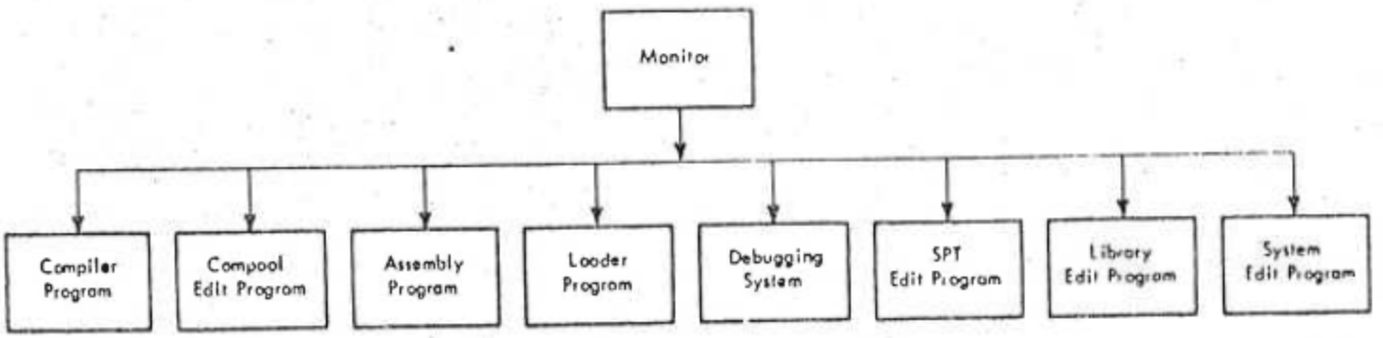

FIGURE 1–1. PROGRAMMING SYSTEM FOR IBM 9020 DATA PROCESSING SYSTEM

The 9020 Library Tape supplied with the IBM 9020 Programming System contains library subroutines that can be used at object time by any JOVIAL program. These subroutines are divided into programmer-called subroutines and compiler-called subroutines. Within these two divisions, the subroutines are further classified according to purpose. In addition to the library subroutines supplied as part of the library, the programmer may write his own JOVIAL library subroutines or BAL programs and add them to the library by using the Library Edit Program. The library subroutines can cause maskable program interruptions. The NOSS Monitor sets the program mask to zero. Programmers who change it should use caution calling library routines.

2.1 PROGRAMMER–CALLED SUBROUTINES

The programmer-called subroutines are those procedures and functions on the library tape that can be called by the programmer in a JOVIAL source program with a procedure or function call. The use of these calls is described in the User's Manual for *IBM 9020 JOVIAL Language.*

The programmer-called subroutines are divided into groups according to purpose. The types of subroutines available to the programmer are mathematical, data comparison, data conversion, and input, output, and input/output unit positioning, checking, initialization and miscellaneous.

The following paragraphs list the types of subroutines available, the name of each subroutine within the type, and its purpose. The individual subroutines are described in greater detail in the following text. In these descriptions, character position refers to the location of a character within a table; character positions are counted from left to right, counting the first as zero. The character position may be considered a byte-displacement addend to the table address.

If a conversion, comparison, or data movement routine requires a table name as an input parameter, the table address is used, and the table declaration is ignored. An item containing an address or an address-type expression such as LOC(TBL) + n may be used instead. The programmer must define the tables properly because the compiler does not check tables used by library subroutines for correctness. The lengths of the individual fields are restricted as indicated for the subroutine being used, but the length of the table depends on the requirements of the JOVIAL program using it. The following JOVIAL statements could be used for a card image table:

```
TABLE    CARD    R   20   $
 BEGIN
  ITEM   COLS    H    4   $
 END
```

Each call has a list of parameters that supply information concerning the type of input, storage locations, quantities involved in the subroutine and type of output. The type of parameters specified in the descriptions of the calls allows for optimal speed of execution. However, the parameters may be in any form, conforming to the rules for using procedures or function calls in the JOVIAL language. The data may also be a different type from that specified, conforming to the rules for mixing data types in the JOVIAL language. (The rules for procedure and function calls and for mixing data are given in the User's Manual for *IBM 9020 JOVIAL Language*.) The parameters are converted by the compiler to the type required, before the library routine is called. Parameters that are not needed for a particular use of the subroutine may be omitted in most cases, but the commas separating the parameters must be present. The last used values of the omitted parameters are not altered by the omission. Some routines cause in-line code to be generated, and omitted arguments are not permitted.

2.1.1 Subroutine Abstracts by Type

Each library subroutine is assigned to one of the following types:

a. Input Data Conversion

b. Output Data Conversion

c. Miscellaneous Data Conversion and Movement

d. Data Comparison

e. I/O – Unit Initialization

f. I/O – Unit Control and Positioning

g. I/O – Input

h. I/O – Data Output

i. I/O – Printing

j. I/O – Card Punching

k. Mathematical

l. Miscellaneous

This subsection describes the programmer-called subroutines.

a. Input Data Conversion

Each input data conversion subroutine accepts as input a description of a character field that is assumed to contain an EBCDIC representation of appropriate data. The subroutine applies its conversion algorithm and returns the converted value as an output argument.

Two different methods of field description are used in these subroutines: direct and indirect. With the direct method, the location and length of the field are included in the calling sequence. With the indirect method, a control table entry contains the location and length of the field. (A scanning procedure, FLTFLD, is available to scan character images for data fields and to store location and length of each field in a control table.) For this method, the calling sequence specifies the location of the control table and the entry that describes the field to be processed.

The subroutines of the input data conversion type are as follows:

ASCIN – Direct: Translates from EBCDIC to ASCII.

FLTFL – Indirect: Converts from EBCDIC floating point to internal floating point.

FLTINJ – Direct: Converts from EBCDIC floating point to internal floating point.

HEXIN – Direct: Converts from EBCDIC hexadecimal to internal binary.

HOLFL – Indirect: Accesses the EBCDIC field without alteration.

HOLINJ – Direct: Accesses the EBCDIC field without alteration.

INTFL – Indirect: Converts from EBCDIC integer to internal binary integer.

INTINJ – Direct: Converts from EBCDIC integer to internal binary integer.

b. Output Data Conversion

Each output data conversion subroutine accepts as input a data value in appropriate internal format and applies its conversion algorithm. The subroutine then deposits the resulting EBCDIC representation in a character field which was described in the procedure call.

The subroutines of the output data conversion type are as follows:

ASCOUT – Translates from ASCII to EBCDIC.

FLTOTJ – Converts from internal floating point to EBCDIC floating point.

HEXOUT – Converts from internal binary to EBCDIC hexadecimal.

HOLOTJ – Deposits the EBCDIC field without alteration.

INTOTZ – Converts from binary integer to EBCDIC integer with leading zeros.

INTOTJ – Converts from internal binary integer to EBCDIC integer.

MIXOTJ – Converts from internal fixed point (scaled to integer) to EBCDIC fixed point.

c. Miscellaneous Data Conversion and Movement

The subroutines of the miscellaneous data conversion and movement type are routines that, though similar, cannot conveniently be classified as input conversion or output conversion:

MVC – Moves data between identically sized character fields (simulates MVC instruction).

TMTR – Executes TRT plus move and TR.

TR – Translates character data from one code to another according to a user-supplied translate table (simulates TR instruction).

TRT – Tests character data by executing a TRT instruction and returns the address of the first nonzero byte.

TSET – Executes a Test and Set instruction and returns a 0 or 1 condition code.

TSTR – Executes TRT and TR in sequence.

LEFTJ – Left justifies Hollerith data that are right justified with leading zeros.

MVI – Moves an immediate character field into an address (simulates MVI instruction).

MVM – Moves a character field of a specified length (which is unlimited) from one address to another address.

MVT – Returns a character field of a specified length from an address as the function output value.

d. Data Comparison

Each data comparison subroutine compares two equal length strings of characters and generates a function output value oriented to capabilities of JOVIAL IF expressions. These subroutines are as follows:

CLC – The arguments specify the locations of the two fields and their length.

CLCI – The arguments specify the location and length of the first field, but include the second field as an argument rather than its address.

e. I/O Unit Initialization

The following I/O unit initialization subroutines should be understood (and in most cases used) before other input/output subroutines are attempted:

SETRET – Provides individual, tailored initialization of specified units.

SYSDTF – Provides initialization of LISTIOs for NOSS processors.

f. I/O Unit Control and Positioning

Each of the following I/O unit control and positioning subroutines calls on the monitor for the appropriate tape control operation:

BSB – Backspaces a physical record on buffered or unbuffered tape.

BSF – Backspaces a file.

BSR – Backspaces a record on unbuffered tape.

REW – Rewinds tape.

RUN – Rewinds and unloads tape.

STF – Forward-spaces a file.

STR – Forward-spaces a record on unbuffered tape.

WEF – Writes end-of-file.

WRE – Writes end-of-file and rewinds.

WRU – Writes end-of-file and rewinds-unloads.

g. I/O Input

The I/O input subroutines enter data from external sources via appropriate monitor routines:

REED – Receives general input from any source.

RTYPE – Reads control messages from the console typewriter.

REEDR – Reads physical records from tape into buffer areas.

VREED – Reads variable blocked records and moves the logical records to a user area.

VREEDR – Reads variable blocked records and passes logical record addresses to the user.

h. I/O Data Output

The following I/O data output subroutines generate output data to any medium but do not provide the special services appropriate to printing and punching:

RYTE – Provides general output to any medium.

i. I/O Printing

The following I/O printing subroutines generate output print images to any medium and automatically provide special services appropriate to printed image requirements:

PRINT – Does general printing, including unblocking and carriage control when required.

TYPE – Types a message on the console typewriter.

TYPEW – Types on console typewriter and delays until operator action.

SPRINT – Does general printing and provides many special functions to simplify page formatting, heading, and numbering.

j. I/O Card Punching

The following I/O card punching subroutines generate output punch images to any medium and provide special services where appropriate for seven-track tape:

PUNCH – Does general punching (including padding for images of less than 80 characters).

The following are special-purpose subroutines used in generating loader format cards:

POBJ – Checksums and punches object deck cards.

PTXT – Formats a block of storage into object deck TXT cards, checksums the cards, and punches them.

PTXNZ – Formats a block of storage into object deck TXT cards, checksums the cards, and punches them, not punching cards for areas containing only zeros.

k. Mathematical

The following mathematical subroutines compute specified mathematical functions of input arguments supplied:

ARCCOS – Computes angle in radians from its cosine.

ARCSIN – Computes angle in radians from its sine.

ARCTAN – Computes angle in radians from its tangent.

ARCTND – Computes heading in degrees from its X and Y components.

ARCTNP – Computes heading in Azimuth Change Pulses (ACPs) from its X and Y components.

ATANFP – Similar to ARCTNP but accepts floating point input.

COS – Computes cosine for angle expressed in radians.

CSINE – Computes sine and cosine for angle expressed in ACPs.

EXPON – Computes EXPON = e(*XX*) for the value XX (computes a number from its natural log).

FSQRT – Computes the square root of a positive number.

LOGN – Computes e(*LOGN*) = XX for the value XX (computes the natural log of XX).

RANDOM – Generates a pseudo random floating point fraction.

RANINT – Generates a pseudo random integer in the range 0.LQ RANINT LS XX, where XX is the input argument.

SIN – Computes sine for angle expressed in radians.

SQRT – Computes square root of a positive number.

l. Miscellaneous

The following are miscellaneous subroutines:

CORE – Dumps selected portions of storage in hexadecimal and EBCDIC.

FLTFLD – Scans a character sequence to locate individual data fields and stores field descriptions in a table for the caller.

GETLIO – Determines the address of the LISTIO entry for a given logical unit.

OPSVC – Provides user-controlled response to Supervisor Calls (SVCs) not recognized by the NOSS Monitor.

SYSCOM – Provides for communication between NOSS processors and the NOSS Monitor.

LOCW – Provides the address of any table or array as a word value.

LTIME – Points the address of the last call to LTIME and the elapsed time in milliseconds since the call.

MODULO – Produces an output that is equivalent to input modulo 14400.

TIMDIF – Accepts an input time and produces its equivalent with absolute value less than 7200.

m. Obsolete Subroutines

The subroutines listed here should not be used because the indicated alternate subroutines are more efficient or easier to use or the listed subroutine is no longer supported. Brief descriptions appear in Appendix A to aid in conversion.

Obsolete Routine	Alternate Routine(s)
ICONVJ	FLTINJ, HEXIN, HOLINJ, INTINJ
FIXOTJ	MIXOTJ
OUTPTJ	FLTOTJ, HEXOUT, HOLOTJ, INTOTZ, INTOTJ, MIXOTJ
MOVKS	MVI
MOVSS	MVC
COMKS	CLCI
COMSS	CLC
USEIO	SETRET
REEDJ	REED OR REEDR
REEDCJ	REED OR REEDR
REEDSI	REED OR REEDR
RYTEJ	RYTE
RYTLIN	PRINT
PRINTJ	PRINT
RYTCD	PUNCH

2.1.2 Routine/Service Characteristics

Table 2–1 lists the routine/service characteristics. The column heads are as follows:

PFX – Prefix to routine name for BAL references.

ROUTINE – Routine name.

N/O/B – Available for code to be run under the NOSS Monitor (N), re-entrant to be run under the Operational Monitor (O), or both (B).

P/F/I – Procedure, function, or implicit type call required.

R/S/I/J – Routine to be called via BAL, service to be called via SVC, implicit conversion routine to be called via nonstandard BAL sequence, or routine which will cause in-line code when called in JOVIAL. (Note that some routines may generate either a call or in-line code.)

D – Indicates parameter list and/or work area must be doubleword aligned.

TS – Amount of temporary storage required at end of argument area in bytes.

TABLE 2–1. ROUTINE/SERVICE CHARACTERISTICS

PFX	Routine	N/O/B	P/F/I	R/S/I/J	D	TS
ZV	ADR	J	F	J		
ZV	ARCCOS	B	F	R	D	64
ZV	ARCSIN	B	F	R	D	72
ZV	ARCTAN	B	F	R	D	56
ZV	ARCTND	B	F	R		4
ZV	ARCTNP	B	F	R		4
ZV*	ASCIN	B	P	R		12
ZV*	ASCOUT	B	P	R		28
ZV	ATANFP	B	F	R		4

*The library will continue to support the older ZW-prefixed version of this routine for a limited time. All new compilations will reference the ZV-version. All BAL references should be changed ASAP.

TABLE 2–1. ROUTINE/SERVICE CHARACTERISTICS (Continued)

PFX	Routine	N/O/B	P/F/I	R/S/I/J	D	TS
ZV*	BSB	N	P	R		
ZV*	BSF	N	P	R		
ZV*	BSR	N	P	R		
ZV	CLC	B	F	R/J		8
ZV	CLCI	B	F	R/J		12
ZV	CORE	N	P	R		0
ZV	COS	B	F	R	D	44
ZV	CSINE	B	P	R		0
ZV	EXPFF	B	I	I	D	
ZV	EXPFI	B	I	I	D	
ZV*	EXPON		F	R		68
ZV	FIX	B	I	I		
ZV	FLTA	B	I	I	D	
ZV*	FLTFL	B	P	R		124
ZV	FLTFLD	B	P	R		40
ZV	FLTI	B	I	I	D	
ZV	FLTINJ	B	P	R	D	80
ZV*	FLTOTJ	B	P	R		96
ZV	FSQRT	B	F	R	D	12
ZV*	GETLIO	N	F	R		
ZV	HEXIN	B	P	R		12
ZV	HEXOUT	B	P	R		8
ZV*	HOLFL	B	P	R		64
ZV*	HOLINJ	B	P	R		20
ZV*	HOLOTJ	B	P	R		28
ZV*	INTFL	B	P	R		52
ZV	INTINJ	B	P	R		12
ZV	INTOTJ	B	P	R		24
ZV	INTOTZ	B	P	R		24
ZV	LEFTJ	B	F	R		0
ZV	LOC	B	F	J		
ZV	LOCW	B	F	R		0
ZV*	LOGN	B	F	R		72
ZV*	LTIME	N	P	R		
ZV	MCSINE	B	P	R		0
ZV	MIXOTJ	B	P	R		32
ZV	MODULO	B	F	R		0
ZV	MVC	B	P	R/J		12
ZV	MVI	B	P	R/J		8
ZV	MVM	B	P	R		8
ZV	MVT	B	F	R		8

*The library will continue to support the older ZW-prefixed version of this routine for a limited time. All new compilations will reference the ZV-version. All BAL references should be changed ASAP.

TABLE 2–1. ROUTINE/SERVICE CHARACTERISTICS (Continued)

PFX	Routine	N/O/B	P/F/I	R/S/I/J	D	TS
ZV	NC	B	P	J		
ZV	NI	B	P	R/J		
ZV	OC	B	P	J		
ZV	OI	B	P	R/J		
ZV*	OPSVC	N	P	R		
ZV*	POBJ	N	P	R		
ZV*	PRINT	N	P	R		
ZV*	PTXNZ	N	P	R		
ZV*	PTXT	N	P	R		
ZV*	PUNCH	N	P	R		
ZW	RANDOM	B	F	R		
ZW	RANINT	B	F	R		
ZV*	REED	N	P	R		
ZV*	REEDR	N	P	R		
ZV*	REW	N	P	R		
ZV*	RTYPE	N	P	R		
ZV*	RUN	N	P	R		
ZV*	RYTE	N	P	R		
ZV*	SETRET	N	P	R		
ZV	SIN	B	F	R	D	44
ZV*	SPRINT	N	P	R		
ZV	SQRT	B	F	R	D	32
ZV*	STF	N	P	R		
ZV*	STR	N	P	R		
ZV*	SYSCOM	N	P	R		
ZV*	SYSDTF	N	P	R		
ZV	TIMDIF	B	F	R		
ZV*	TMTR	B	P	R		24
ZV	TR	B	P	R		8
ZV*	TRT	B	P	R		16
ZV	TSET	B	F	R		0
ZV*	TSTR	B	P	R		20
ZV*	TYPE	N	P	R		
ZV*	TYPEW	N	P	R		
ZW	VREED	N	P	R		
ZW	VREEDR	N	P	R		
ZV*	WEF	N	P	R		
ZV*	WRE	N	P	R		
ZV*	WRU	N	P	R		
ZV	XC	B	P	J		
ZV	XI	B	P	R/J		

*The library will continue to support the older ZW-prefixed version of this routine for a limited time. All new compilations will reference the ZV-version. All BAL references should be changed ASAP.

2.1.3 BAL Programming Considerations in Using Library Facilities

Library facilities are organized with the specific requirement of servicing calls generated by the JOVIAL compiler for JOVIAL procedure call statements and function call expressions. To fulfill these requirements, all subroutines have certain characteristics in common. The following should be considered applicable to all subroutines except where specific exceptions are provided in individual subroutine descriptions:

a. Functions have exactly the same characteristics as procedures with the same number of input arguments and a single output argument.

b. Each argument is expected to be alone and right justified in a fullword field or (if necessary) in a two-word field (not necessarily on a doubleword boundary). These fields are grouped together, in the same sequence as the argument fields in a procedure call statement, in an out-of-line argument area. For some re-entrant routines and services, a special "temporary storage" field is defined and must be provided at the end of the argument area. This is indicated by a "BAL save area" size following the parameters for the routine.

c. The parameter list and/or work area is normally expected to be fullword aligned. Where doubleword alignment is needed, it is indicated both in Table 2–1 and in the individual routine description.

d. Before the routine or service is called, each input argument must be placed in the argument area in proper format.

   General registers 2 through 13 and floating point registers 2 and 4 are always returned exactly as found. Register 14 is normally returned as found, but must be loaded in the routine call. General registers 0, 1 and 15 and floating point registers 0 and 6 have unpredictable contents on return. In some cases register 14 may be updated by 4 prior to return.

e. Actual calls are best described by example. The examples below are for a routine named RTN and its out-of-line argument area ARG.

   1. A re-entrant format routine with arguments:

```
LA      10,ARG
L       15,=A(ZVRTN)
BALR    14,15
```

   2. A service with arguments and a code of 57:

```
LA      10,ARG
SVC     57
```

   3. A service with no arguments and a code of 62:

```
SVC     62
```

4. A re-entrant format routine with one input and one output argument, both assigned to registers:

```
L       10, INARG
L       15, = A(ZVRTN)
BALR    14, 15
ST      1, OUTARG
```

(Register 1 may be used when there is a single output argument. Register 10 may be used in this manner when there is one input argument, no temp storage requirement, and either no output argument or a single output argument in register 1.)

5. A routine with arguments and a displacement of 12:

```
L       15, = A(ZWRTN)
CNOP    0, 4
BAL     14, 12(0, 15)
DC      A(ARG)
```

NOTE: This format is obsolete and is being phased out.

f. Return immediately follows the BALR instruction, or the address constant (ADCON) if present, or (where appropriate) the SVC. At this point, output data is in the argument area ready to be processed (or in register 1).

g. On return from a function, the output value is stored in the word (or two words if needed) immediately following the last input parameter. Function outputs are flagged with "(O/P)" in the parameter descriptions.

2.1.4 Subroutine Descriptions

This subsection describes the programmer-called routines.

2.1.4.1 ADR Function

Purpose: The ADR function allows the JOVIAL user access to the actual address of a JOVIAL-declared data element.

Use: See the JOVIAL User's Manual for a description of the uses and restrictions.

NOTE: This routine is on the library solely to permit the JOVIAL compiler to produce appropriate in-line code.

2.1.4.2 *ARCCOS Function*

Purpose: The ARCCOS function computes an angle expressed in radians from its cosine.

Input: The input is the cosine of an angle. ARCCOS is re-entrant.

Output: The output is an angle expressed in radians, given in normalized floating-point format.

JOVIAL Function Call:

ARCCOS(xxx)

where:

xxx is the input argument.

Range: $-1 \leqslant$ cosine $\leqslant 1$ (No validity test is made of the cosine; negative cosines are computed in the second quadrant.)

$0 \leqslant$ angle $\leqslant \pi$, expressed in radians.

Accuracy: For angles greater than 1, the error does not exceed ±4.8 in the seventh significant figure. For angles less than 1, the absolute error does not exceed 4.8×10^{-7}, except for angles whose cosines approach 1, as follows:

Cosine	Error in Angle
.999	-2×10^{-7}
.9999	$+1 \times 10^{-6}$
.99999	$+3 \times 10^{-6}$
.999999	-9×10^{-6}
.9999999	-4×10^{-5}

BAL Calling Sequence:

```
        LA      10, AJ1
        L       15, = A(ZVARCCOS)
        BALR    14, 15
AJ1     DS      0D      Note: Doubleword
                        alignment is required.
AJARG   DS      F       Input Argument
AJOUT   DS      F       Output Area
        DS      64C     Save/Work Area
```

2.1.4.3 *ARCSIN Function*

Purpose: The ARCSIN function computes an angle expressed in radians from its sine.

Input: The input is the sine of an angle.

Output: The output is an angle expressed in radians, given in normalized floating-point format.

JOVIAL Function Call:

ARCSIN(xxx)

where:

xxx is the input argument.

BAL Calling Sequence:

```
        L       10, EB1
        L       15, = A(ZVARCSIN)
        BAL     14, 15
EB1     DS      0D      Note: Doubleword
                        alignment is required.
EBARG   DS      F       Input Argument
EBOUT   DS      F       Output Area
        DS      72C     Save/Work Area
```

Range: $-1 \leqslant$ sine $1 \leqslant$ (No validity test is made of the sine; negative sines are computes in the fourth quadrant.)

$-\frac{\pi}{2} \leqslant$ angle $\leqslant \frac{\pi}{2}$, expressed in radians.

Accuracy: For angles less than 1, the absolute error does not exceed +1 binary place. For angles greater than 1, the error does not exceed ±4.8 in the seventh significant figure, except for angles whose sines approach 1, as follows:

Sine	Error in Angle
.999	$+2 \times 10^{-7}$
.9999	-1×10^{-6}
.99999	-3×10^{-6}
.999999	$+9 \times 10^{-6}$
.9999999	$+4 \times 10^{-5}$

2.1.4.4 ARC Tangent (ARCTAN) Function

Function Call Format: ARCTAN(TAN)

Purpose: ARCTAN computes an angle expressed in radians from its tangent.

Use: The argument to this routine is: TAN is the tangent of an angle. ARCTAN is re-entrant.

Restrictions:

The range of input is $-.72370051 \times 10^{76} \leq \text{Tangent} \leq .82370051 \times 10^{76}$

The range of output is $-\frac{\pi}{2} < \text{Angle} < \frac{\pi}{2}$

The absolute error does not exceed 4.8×10^{-7}.

Parameters:

ITEM	TAN	F	$
ITEM	ARCTAN	F	$ (O/P)

BAL save area: 56 bytes

Doubleword alignment is required.

Length: 568 decimal bytes.

2.1.4.5 Arc Tangent in Degrees (ARCTND) Function

Function Call Format: ARCTND(ARGA, ARGB)

Purpose: ARCTND converts a vector presented in terms of X and Y displacement to a heading or angle expressed in degrees.

Use: ARGA is the component that will be positive at 0 degrees and negative at 180 degrees. ARGB is the component that will be positive at 90 degrees and negative at 270 degrees. This routine compares ARGA and ARGB to determine the 45-degree segment which contains the angle, computes position of the angle within the segment, rounds to the nearest degree, adjusts for segment position and returns the result (0 through 360). ARCTND is re-entrant.

Restrictions: ARGA and ARGB magnitudes are limited to 2(*24*) −1 (16, 777, 215). In case of input error, leading bits will be ignored. If ARGA and ARGB are both zero, ARCTND will be zero.

Parameters:

ITEM	ARGA	I	32	S	$	
ITEM	ARGB	I	32	S	$	
ITEM	ARCTND	I	32	S	$	(O/P)

BAL save area: 4 bytes

Length: 200 decimal bytes

Execution Time: 350 microseconds

2.1.4.6 Arctangent in ACPs (ARCTNP, ATANFP). There are two functions for arctangents in ACPs. One is for fixed point input; the other for floating point input.

a. Arctangent in ACPs (ARCTNP) Function – Fixed point input

Function Call Format: ARCTNP(ARGA, ARGB)

Purpose: ARCTNP converts a vector presented in terms of X and Y displacement to a heading or angle expressed in ACPs.

Use: ARGA is the component that will be positive at 0 degrees and negative at 180 degrees. ARGB is the component that will be positive at 90 degrees and negative at 270 degrees. This routine compares ARGA and ARGB to determine the 45-degree segment which contains the angle, computes position of the angle within the segment, rounds to the nearest ACP, adjusts for segment position and returns the result (0 through 4096). ARCTNP is re-entrant.

Restrictions: ARGA and ARGB magnitudes are limited to 2(*24*) −1 (16, 777,215). In case of input error, leading bits will be ignored. If ARGA and ARGB are both zero, ARCTNP will be zero.

Parameters:

```
ITEM   ARGA      I   32   S   $
ITEM   ARGB      I   32   S   $
ITEM   ARCTNP    I   32   S   $   (O/P)
BAL save area: 4 bytes
```

Length: 200 decimal bytes

Execution Time: 350 microseconds

b. Arctangent in ACPs (ATANFP) Function -- Floating-point Input

Function Call Format: ATANFP(ARGA, ARGB)

Purpose: ATANFP converts a vector presented in terms of X and Y displacement to a heading of angle expressed in ACPs.

Use: ARGA is the component that will be positive at 0 degrees and negative at 180 degrees. ARGB is the component that will be positive at 90 degrees and negative at 270 degrees. This routine compares ARGA and ARGB to determine the 45-degree segment which contains the angle, computes position of the angle within the segment, rounds to the nearest ACP, adjusts for segment position and returns the result (0 through 4096). ATANFP is re-entrant.

Restrictions: If both ARGA and ARGB are zero, ATANFP will be zero.

Parameters:

```
ITEM   ARGA      F   $
ITEM   ARGB      F   $
ITEM   ATANFP    I   32   S   $   (O/P)
BAL save area: 4 bytes
```

Length: 184 decimal bytes

Execution Time: 300 microseconds

2.1.4.7 ASCIN Procedure. The ASCIN procedure converts EBCDIC characters to ASCII code representation. The maximum number of characters that can be converted is eight. If the number of characters to be converted is greater than eight, the leftmost characters are truncated; if the number of characters is less than eight, leading blanks are added after the conversion. The procedure call for ASCIN is:

```
ASCIN(Table, First, Number = Item)  $
```

where:

Table is the name of a table of EBCDIC characters.

First is an unsigned integer item or constant that specifies the character position of the first character to be converted.

Number is an unsigned integer item or constant that specifies the number of characters to be converted.

Item is an ASCII item in which the converted characters will be placed.

2.1.4.8 ASCOUT Procedure. The ASCOUT procedure converts an ASCII item from one to eight characters in length to EBCDIC and places the converted item in a programmer-defined output area. If the number of character positions in the output area is greater than the number of characters to be converted, the characters are right-justified and leading blanks are inserted; if the number of character positions in the output area is less than the number of characters to be converted, the left most characters are truncated. The procedure call for ASCOUT is:

ASCOUT(Item, Table, First, Number) $

where:

Item is the ASCII item to be converted.

Table is the name of the table into which the converted item is to be placed.

First is an unsigned integer item or constant that specifies the first character position within the output table to be used as the output area.

Number is an unsigned integer item or constant that specifies the total number of character positions in the output area.

2.1.4.9 Backspace Block (BSB) Procedure

Procedure Call Format: BSB(UNIT = CHECK)$

Purpose: BSB backspaces one physical record on buffered or unbuffered tapes.

Use: The argument to this routine is:

UNIT -- The logical unit number of the tape.

CHECK – The check character to indicate success and failure is as follows, assuming the programmer has properly initialized error and end-of-file returns for the tape unit being used:

0 -- Success

1 -- The physical record was an end-of-file

2 -- An uncorrectable error was encountered

This routine effectively backspaces one physical record. To do this, it actually backspaces two physical records if the previous operation was a buffered read. The normal use of this routine is to position buffered tapes that cannot be positioned via procedure BSR.

Restrictions: Tape reading is by logical records. BSB backspaces a physical record which may represent several logical records.

If the check character is to be used, the appropriate entry to SETRET must be made before the call to BSB. BSB can only be used with the Utility Monitor.

Parameters:

ITEM	UNIT	I	32	S	$	(Reg 10)
ITEM	CHECK	I	32	S	$	(Reg 1)

Services Used: SYSCTL

Length: 40 decimal bytes

2.1.4.10 BSF Procedure. The BSF procedure backspaces a file on the selected tape. When the operation is complete, the tape will be positioned to the left of the tapemark that terminated the backspace operation. The procedure call for BSF is:

```
BSF(unit) $      (UNIT passed in Reg 10)
```

2.1.4.11 BSR Procedure. The BSR procedure backspaces a record on the selected tape and checks for an end-of-file condition. When the operation is complete, the tape will be positioned so that the first word of the backspaced record can be read. A backspace record command given when the tape is at load point results in no operation; a backspace record command given for a buffered unit or an uncorrectable error results in termination of the job. The procedure call for BSR is:

```
BSR(unit = check) $      (UNIT in Reg 10, CHECK in Reg 1)
```

where:

Check is a 32-bit unsigned integer item that will be set to 1 if an end-of-file condition is sensed during the operation and 0 if no end-of-file is sensed.

2.1.4.12 Compare Logical Character (CLC) Function

Function Call Format: CLC(ADRA, LTH, ADRB)

Purpose: CLC compares two character fields and determines whether the first field is greater than, equal to or less than the second.

Use: The arguments to this routine are:

ADRA – The address (name of the table) of the first field to be compared.

LTH – The length of the field in bytes.

ADRB The address (name of the table) of the second field to be compared.

This routine compares the two fields in question byte by byte from left to r ight. If the routine compares across the entire field and all characters are found equal, CLC takes the value 0. If an unequal comparison is found in the field, the routine does not compare further and takes the value 1 if the unequal character of the first field is greater than the associated character of the second field. If the unequal character of the first field is less than the associated character of the second field, CLC takes the value –1.

A common use of this function is the comparison of two lists of EBCDIC names. The programmer might use the following code:

```
IF CLC(NAME1, 12, NAME2) GQ 0 $ GOTO GREQ $
```

so that, if the 12 bytes beginning at address NAME1 are greater than or equal to the 12 bytes beginning at address NAME2, program control will pass to the statement labeled GREQ.

The relational operator (GR, EQ,...) should express the relationship between ADRA and ADRB, so the programmer should always compare against zero. CLC is re-entrant.

Parameters:

ITEM	ADRA	I	32	S	$	
ITEM	LTH	I	32	S	$	
ITEM	ADRB	I	32	S	$	
ITEM	CLC	I	32	S	$	(O/P)

BAL save area: 8 bytes

Length: 96 decimal bytes

NOTE: This routine may cause generation of in-line code; therefore no omitted arguments are allowed.

2.1.4.13 Compare Logical Character Immediate (CLCI) Function

Function Call Format: CLCI(ADRA, LTH, VAL)

Purpose: CLCI compares a character field with an immediate character field and determines whether the first is greater than, equal to or less than the second.

Use: The arguments to this routine are:

ADRA – The address (name of the table) of the first field to be compared

LTH – The length of the field in bytes

VAL – The field to be compared

This routine compares the two fields in question byte by byte from left to right. If all the characters are found to be equal, CLCI takes the value 0.

If an unequal comparison is found in the fields, and if the unequal character of the first field is greater than the associated character of the immediate field, CLCI takes the value 1.

If the unequal character of the first field is less than the associated character of the immediate field, then CLCI takes the value –1.

A typical use of this function is to compare a symbol from the header of a message with the task code to see whether the program is responsible for handling the message. The programmer might use the following code:

```
IF CLCI(MSG1, 5, 5H(SCOPE)) EQ 0 $
GOTO MYPROG $
```

so that, if the first five characters of the message were 5H(SCOPE), program control would pass to the statement labeled MYPROG.

The relational operator (GR, GQ, EQ,...) should express the relationship between ADRA and VAL, so the programmer should always compare against zero. CLCI is re-entrant.

Restrictions: The length provided as the second argument of the function must lie in the range $1 \leqslant LTH \leqslant 8$. The third argument will be used as is only if it is unsigned integer/Hollerith. Other data types will be converted to this format by JOVIAL.

Parameters:

ITEM	ADRA	I	32	S	$
ITEM	LTH	I	32	S	$
ITEM	VAL	H	8	$	$
ITEM	CLCI	I	32	S	$

BAL save area: 12 bytes

Length: 56 decimal bytes

NOTE: This routine may cause generation of in-line code; therefore no omitted arguments are allowed.

2.1.4.14 Core Dump in Hexadecimal (CORE) Procedure

Procedure Call Format: CORE(LOC, NUMB, NAME, WORDS)$

Purpose: CORE prints the contents of a specified number of words in hexadecimal on the system output unit.

CORE includes an alphanumeric translation for each line dumped in hexadecimal. Non-printable characters will be changed to a(.). The alphanumeric translation appears to the right of the Hex dump and is enclosed by a pair of asterisks. Relative addresses appear on the extreme right.

Example:

```
CORE(RLTB, 15, 4H(RLTB), 3)$
```

where:

Table RLTB is at core location X'19000'.

RLTB

019000	40C1C2C3	C440001D	00000070	* ABCD . . . . . *	000000
01900C	40C1C2C3	C440001B	00000170	* ABC . . . . . *	00000C
019018	40C1C2C3	C440001C	00000270	* ABCD . . . . . *	000018
019024	40E3C5E2	E340001E	00000070	* TEST . . . . . *	000024
019030	00000000	00000000	00000000	* . . . . . *	000030

Use: The arguments to this routine are:

LOC – The address of the first word to be dumped.

NUMB – The number of words to be dumped; if zero, only the name of the dump is printed.

NAME -- The label for the dump; e.g., 8H(COREDUMP).

WORDS – The number of words per entry. The magnitude of WORDS is used in formatting the output. This specified number of words will be printed on one or more lines, with eight words per line and any remainder on the last line. The next entry will be started on a new line. The final entry will be shorter than other entries if NUMB is not an exact multiple of WORDS.

If WORDS is positive, this dump will start on a new page; if negative, two lines will be skipped after any previous printing and CORE will restore to a new page only when the cumulative total of its printing indicates this is appropriate.

Restrictions: CORE is formatted as a re-entrant routine. It is not really re-entrant, but since it can only run under the NOSS monitor, no problems are anticipated.

Parameters:

ITEM	LOC	I	32	S	$
ITEM	NUMB	I	32	S	$
ITEM	NAME	H	8		$
ITEM	WORDS	I	32	S	$

Length: 1128 decimal bytes

2.1.4.15 Cosine (COS) Function

Function Call Format: COS(THETA)

Purpose: COS computes the cosine of an angle expressed in radians.

Use: The argument to this routine is THETA, the angle in radians. COS is re-entrant.

Restrictions: COS(THETA) is computed as SIN(THETA + PI/2).

Parameters:

ITEM	THETA	F	$	
ITEM	COS	F	$	(O/P)

BAL save area: 44 bytes

Doubleword alignment is required.

Services Used: SIN

Length: 32 decimal bytes

2.1.4.16 *Compute SIN/COS from ACPs (CSINE) Procedure*

Procedure Call Format: CSINE(ACPS = SIN, COS)$

Purpose: This routine finds the SIN and COS of an angle stated in ACPs.

Use: The arguments to this routine are:

ACPS – The input angle in ACPs (360° = 4096 ACPs).

SIN – The sine of the angle to 14-bit fraction accuracy.

COS – The cosine of the angle to 14-bit fraction accuracy.

The routine uses a table lookup. It is designed primarily for speed and for space economy, which explains the limited accuracy. The table may be accessed directly via entry point ZVSINCOS. It consists of 1024 HS14 constants.

This routine probably will be called often from BAL or DIRECT code. The BAL calling sequence is:

```
ST      REG, DACP       DACP is a three-word area
LA      10, DACP
L       15, = A(ZVCSINE)
BALR    14, 15
```

On return, the SIN is in register 0 as well as DACP+4, and the COS is in register 1 as well as DACP+8. To obtain maximum efficiency, the programmer should arrange his storage to conform with the format given here. This will avoid moving the data. CSINE is re-entrant.

Parameters:

```
ITEM  ACPS   I  32  S      $
ITEM  SIN    A  32  S  14  $
ITEM  COS    A  32  S  14  $
```

Length: 2112 decimal bytes

Execution Time: 95 microseconds

2.1.4.17 *EXPON Function*

Purpose: The EXPON function computes the value of $e^{(input)}$.

Input: The input is the power to which e is raised.

Output: The output is given in normalized floating-point format.

JOVIAL Function Call:

EXPON(xxx)

where:

xxx is the input argument.

BAL Calling Sequence:

```
LA      10, Pararea
L       15, = A(ZVEXPON)
BALR    14, 15
```

where:

Pararea is the address of an 8-byte parameter area that contains a normalized floating-point number in the first four bytes and the result in the last four bytes, followed by a 17 word working storage area. Pararea must be doubleword aligned.

Range: $-180.21826 < \text{input} < +174.67308$ (if the input is less than -180.21826, a zero is returned; if the input is greater than +174.67308, the input number is returned.)

Accuracy: The error does not exceed ±4.8 in the seventh significant figure.

2.1.4.18 Convert EBCDIC to Internal Floating Point (FLTFL) Procedure

Procedure Call Format: FLTFL(TBL, FLD, FLT)

Purpose: FLTFL converts an EBCDIC decimal integer of not more than 10 digits, with an optional sign and exponent, to its floating-point representation.

Use: The arguments to this routine are:

TBL – The address of the field description table as defined in the FLTFLD procedure.

FLD – The entry number in the field description table.

FLT – The floating-point output item.

Before the FLTFL procedure is called, the FLTFLD procedure should be used to set up the field description table (TBL). FLTFL sets up the parameters necessary to call ZVFLTINJ, which converts an EBCDIC decimal integer to its floating-point representation.

Restrictions: Same as for FLTINJ.

Parameters:

```
ITEM   TBL     I   32   S   $
ITEM   FLD     I   32   S   $
ITEM   FLT     F            $
BAL save area: 31 words
```

Services Used: ZVFLTINJ

2.1.4.19 Floating Field (FLTFLD) Procedure

Procedure Call Format: FLTFLD(TTBL, TBL, FIRST, LTH) $

Purpose: FLTFLD scans an input stream and determines the position and length of data fields within the stream.

Use: The arguments to this routine are:

TTBL – The address of the user's translate table, the contents of which are defined below.

TBL – The address of the field description table in which to place information about the input stream.

FIRST – The address of the beginning byte of the input stream.

LTH – The length of the input stream in bytes.

The Translate Table (TTBL) is 64 words long (256 bytes) and is used with the Translate and Test (TRT) instruction. Each control character in the table is associated with the EBCDIC character whose hexadecimal value is equal to the byte address (relative to the beginning of the table) of the control character. The control characters are:

0 – If the associated EBCDIC character can legally appear in a data field.

1 – If the associated EBCDIC character is a field separator (delimiter) whose position in the input string is of no concern.

2 – If the associated EBCDIC character is a field separator (delimiter) whose position in the input string is of concern.

3 – If the associated EBCDIC character is a field separator (delimiter) used to enclose a comments field or some parenthetical field.

The field description table is composed of two-word entries. Entry zero is a control entry

where:

FLDA is set to the number of fields FLTFLD found in the input string.

FLDL is the number of entries in the field description table exclusive of entry zero. This word of entry zero must be preset by the user.

Each following entry is set by FLTFLD in sequence as it finds data fields:

FLDA is the address of the starting byte of the data field.

FLDL is the length of the field.

This routine scans the input string and generates entries in TBL that describe the data fields. It uses the TRT instruction and Table TTBL to find the delimiters. FLTFLD ignores type 1 delimiters unless they follow a data field. When a data field is present, the position of the type 1 delimiter is used by the routine to determine the field length and the appropriate entry is placed in TBL describing the data field. When the routine encounters a type 2 delimiter preceded by a data field, it places an entry in TBL describing the data field and places a consecutive entry in TBL describing the type 2 entry delimiter. FLTFLD makes a single entry in TBL for a type 2 delimiter when it is not preceded by a data field.

When the routine encounters a type 3 delimiter, it treats the information between that delimiter and the next type 3 delimiter, or between that delimiter and the end of the input string, as parenthetical information, regardless of embedded type 1 and type 2 delimiters. Type 3 delimiters do not cause generation of field description table entries. The routine scans the input string until the string is exhausted or until there are no more available entries in the Field Description Table (TBL). Then FLTFLD returns control to the user, with TBL containing the following information:

FLDA of entry 0 contains the count of the available entries used (the number used not counting entry 0).

FLDL of entry 0 remains unchanged.

For the remaining entries that describe data fields,

FLDA of the entry is the starting address of the field.

FLDL of the entry is the fields length.

For the remaining entries that describe type 2 delimiters,

FLDA is the address of the delimiter.

FLDL is a zero.

Suppose that the Translate Table is coded so that the relationship shown below exists.

EBCDIC Character		Translate Table	
Character	Hexadecimal	Byte Address	Hexadecimal
Space	40	40	01
$	5B	5B	02
,	6B	6B	01
A – I	C1 – C9	C1 – C9	00 – 00
J – R	D1 – D9	D1 – D9	00 – 00
S – Z	E2 – E9	E2 – E9	00 – 00
0 – 9	F0 – F9	F0 – F9	00 – 00
		All others	02

Then the user might code:

```
.
.
FLTFLD (TRANS, FLDS, CARD, 60) $
IF FLDL ($1$) EQ 0 $ GOTO CNTRL $
DATA. FOR N = 1, 1, FLDA $
MVC (FLDA($N$), FLDL($N$), LOC(ARGLST)+2*N) $
.
.
```

Restrictions: It is possible that less than the entire input stream would be scanned. This would occur if there were more fields than field description table entries. The remainder of the input stream could be scanned by setting

```
LTH = LOC(CARD)+LTH–ADRA($N$)–ADRL($N$) $
```

and calling

```
FLTFLD(TTBL, TBL, ADRA($N$)+ADRL($N$),LTH) $
```

where:

ADRA(N) and ADRL(N) are the items of the last entry in the field description table.

Parameters:

```
ITEM   TTBL      I   32    S   $
ITEM   TBL       I   32    S   $
ITEM   FIRST     I   32    S   $
ITEM   LTH       I   32    S   $
BAL save area:  40 bytes
```

Table Formats:

```
TABLE TTBL R 1 64 $
   BEGIN
      STRING COL H 1 0 0 B 1 4 $
   END

TABLE TBL R n+1  "WHERE n IS THE MAXIMUM
                  NUMBER OF FIELDS
                  DESIRED" 2 $
   BEGIN
      ITEM FLDA I 32 S Ø  Ø  N     $
      ITEM FLDL I 32 S 1  Ø  N     $    BEGIN n END
   END
```

Length: 200 decimal bytes

2.1.4.20 Convert an EBCDIC Decimal Number to Floating Point (FLTINJ) Procedure

Procedure Call Format: FLTINJ(TABLE, FIRST, NUMBER = ITEM) $.

Purpose: The routine converts a decimal number in EBCDIC representation to an internal floating-point representation.

Use: The arguments to this routine are:

TABLE – The name of the table that contains EBCDIC characters.

FIRST – The position of the first character to be converted.

NUMBER – The number of characters to convert.

ITEM – Floating-point output item.

Restrictions: Conversion is not possible, and binary zeros are returned as output if the routine encounters invalid characters, a number greater than nine digits in length, an input field greater than 15 bytes in length, or a number less than $.53976054 \times 10^{-78}$. If a number is greater than $.72370051 \times 10^{76}$, this maximum number will be moved to the output area.

If an explicit sign is present, it must be in the first column of the input field.

Parameters:

ITEM	TABLE	I	32	S	$
ITEM	FIRST	I	32	S	$
ITEM	NUMBER	I	32	S	$
ITEM	ITEM	F			$

BAL save area: 80 bytes

Doubleword alignment is required.

Length: 968 decimal bytes

2.1.4.21 FLTOTJ Procedure. The FLTOTJ procedure converts a floating-point number in internal format to EBCDIC decimal and places it, right-justified, in a programmer-defined output area. The final representation in EBCDIC contains a sign, a decimal point, a fraction, E, and a signed exponent. The procedure call for FLTOTJ is:

FLTOTJ(Item, Table, First, Number) $

where:

Item designates the floating-point item to be converted.

Table is the name of the table in which the converted number is to be placed.

First is an unsigned integer item or constant that specifies the first character position of the table to be used as the output area.

Number is an unsigned integer item or constant that specifies the total number of character positions to be used as the output area; the number must include provision for the sign, decimal point, E, and signed exponent. It must be greater than 6 and less than or equal to 15.

2.1.4.22 Fortran Square Root (FSQRT) Function

Function Call Format: FSQRT(VAL)

Purpose: FSQRT computes the square root of a positive number. It is intended for use in place of SQRT where core space and/or execution time are more important than extreme accuracy. Truncation error may occur in the low order bit.

Use: The input argument to this routine is VAL, the value to be processed. FSQRT is re-entrant.

Restrictions: If a negative argument is supplied, FSQRT will return the square root of its absolute value.

Parameters:

ITEM	VAL	F	$	
ITEM	SQRT	F	$	(O/P)

BAL save area: 12 bytes

Doubleword alignment is required.

Length: 152 bytes

Execution Time: Approximately one-half the time required by SQRT.

2.1.4.23 Get LISTIO Address (GETLIO) Function

Function Call Format: GETLIO(LU)

Purpose: GETLIO provides a means of obtaining the location of the LISTIO entry (the table that contains control and status information about I/O devices) for a particular logical I/O unit.

Use: The argument to this routine is LU, the logical unit number of the device whose LISTIO entry is to be found. The routine compares the logical unit number presented to it with the logical unit numbers in the unit assignment table. If it finds a match, GETLIO takes the value of the address of the unit's LISTIO entry. If the routine does not find a matching logical unit number in the unit assignment table, GETLIO takes the value zero.

A common use of this function is to determine the device address of a logical unit. The programmer might code:

```
LIO5 - GETLIO (5) $
IF LIO5 EQ 0 $ GOTO ERR $
MVC (DEVAD +2, 2, LIO5 + 62) $
```

so that, if there is a logical device 5, the device address will be placed in DEVAD.

Parameters:

ITEM	LU	I	32	S	$		(Reg 10)
ITEM	GETLIO	I	32	S	$	(O/P)	(Reg 1)

Length: 56 decimal bytes

2.1.4.24 Hexadecimal Input Conversion (HEXIN) Procedure

Procedure Call Format: HEXIN(ADR, COL, LTH = DATA) $

Purpose: HEXIN converts a hexadecimal number represented in EBCDIC to an unsigned 32-bit number.

Use: The arguments to this routine are:

ADR plus COL -- The address of the first character to be converted.

LTH -- The number of characters to be converted.

DATA -- The converted value.

HEXIN is re-entrant.

Restrictions: Output is limited to 32 bits (8 digits). Illegal characters will result in unpredictable output.

Parameters:

ITEM	ADR	I	32	S	$
ITEM	COL	I	32	S	$
ITEM	LTH	I	32	S	$
ITEM	DATA	I	32	U	$

BAL save area: 12 bytes

Length: 72 decimal bytes

2.1.4.25 Hexadecimal Output Conversion (HEXOUT) Procedure

Procedure Call Format: HEXOUT(DATA, ADR, COL, LTH)$

Purpose: HEXOUT converts an unsigned integer to EBCDIC representation of its hexadecimal equivalent.

Use: The arguments to this routine are:

DATA – The integer to be converted.

ADR plus COL – The address of the first character of the output area.

LTH – The number of characters in the output area.

HEXOUT is re-entrant.

Restrictions: Input is limited to 32 bits (8 characters).

Parameters:

ITEM	DATA	I	32	U	$
ITEM	ADR	I	32	S	$
ITEM	COL	I	32	S	$
ITEM	LTH	I	32	S	$

BAL save area: 8 bytes

Length: 72 decimal bytes

2.1.4.26 Process Floating Fields (HOLFL) Procedure

Procedure Call Format: HOLFL(TBL, FLD = HOL)

Purpose: HOLFL moves a maximum of eight EBCDIC characters from a specified area to a storage location.

Use: The arguments to this routine are:

TBL – The address of the field description table as defined in the FLTFLD procedure.

FLD – The entry number in the field description table.

HOL – An EBCDIC field in which the characters are to be placed.

Before the HOLFL procedure is called, the FLTFLD procedure should be used to set up the field description table (TBL). HOLFL sets up the parameters necessary to call ZWHOLINJ, which moves a maximum of eight EBCDIC characters from a specified area to an EBCDIC item.

Parameters:

ITEM	TBL	I	32	S	$
ITEM	FLD	I	32	S	$
ITEM	HOL	H	8	$	

BAL save area: 16 words

Services Used: ZVHOLINJ

Length: 64 decimal bytes

2.1.4.27 HOLINJ Procedure. The HOLINJ procedure moves a maximum of eight EBCDIC characters from a specified area to an EBCDIC item. If the number of characters is more than eight, the leftmost characters are truncated; if the number of characters is less than eight, the item is filled with leading zeros. The procedure call for HOLINJ is:

```
HOLINJ(Table, First, Number = Item) $
```

where:

Table is the name of a table of EBCDIC characters.

First is an unsigned integer item or constant that specifies the first character to be moved.

Number is an unsigned integer item or constant that specifies the number of characters to be moved.

Item designates an EBCDIC item in which the characters are to be placed.

2.1.4.28 HOLOTJ Procedure. The HOLOTJ procedure moves from one to eight EBCDIC characters to a programmer-defined output area. If the number of character positions in the output area is greater than the number of characters to be moved, the characters are right-justified and leading blanks are supplied; if the number of character positions in the output area is less than the number of characters to be moved, the leftmost characters are truncated. The procedure call for HOLOTJ is:

```
HOLOTJ(Item, Table, First, Number) $
```

where:

Item designates the EBCDIC item to be moved.

Table is the name of the table into which the item is to be moved.

First is an unsigned integer item or constant that specifies the first character position of the table to be used as the output area.

Number is an unsigned integer item or constant that specifies the total number of character positions to be used as the output area.

2.1.4.29 *Convert EBCDIC to Internal Binary (INTFL) Procedure*

Procedure Call Format: INTFL(TBL, FLD = INT)

Purpose: INTFL converts an EBCDIC decimal integer of not more than 10 digits, with an optional sign and exponent, to its fixed-point representation (a signed binary integer).

Use: The arguments to this routine are:

TBL – The address of the field description table as defined in the FLTFLD procedure.

FLD – The entry number in the field description table.

INT – The integer output item.

Before the INTFL procedure is called, the FLTFLD procedure should be used to set up the field description table (TBL). INTFL sets up the parameters necessary to call ZVINTINJ, which converts an EBCDIC decimal integer to its fixed-point representation.

Restrictions: Same as for INTINJ.

Parameters:

ITEM	TBL	I	32	S	$
ITEM	FLD	I	32	S	$
ITEM	INT	I	32	S	$

BAL save area: 13 words

Services Used: ZVINTINJ

Length: 72 decimal bytes

2.1.4.30 *Convert EBCDIC integer to a Signed Binary Number (INTINJ) Procedure*

Procedure Call Format: INTINJ(TABLE, FIRST, NUMBER = ITEM) $

Purpose: This routine converts an EBCDIC decimal integer of not more than 10 digits, with an optional sign and exponent, to its fixed-point representation (a signed binary integer).

Use: The arguments to this routine are:

TABLE – The name of the table that contains EBCDIC characters.

FIRST – The position of the first character to be converted.

NUMBER – The number of characters to convert.

ITEM – Integer output item.

INTINJ is re-entrant.

Restrictions: Conversion is not possible and binary zeros are returned as output if the routine encounters invalid characters, a number greater than 10 digits in length, an input field greater than 15 bytes in length, or a number exceeding the maximum allowed values: $2,147,483,647 \geq \text{integer} \geq -2,147,483,648$.

If an input value is explicitly signed, the sign must immediately precede the first digit.

Parameters:

ITEM	TABLE	I	32	S
ITEM	FIRST	I	32	S
ITEM	NUMBER	I	32	S
ITEM	ITEM	I	32	S

BAL save area: 12 bytes

Length: 304 decimal bytes

2.1.4.31 Convert Integer Item to EBCDIC with Leading Blanks (INTOTJ) Procedure

Procedure Call Format: INTOTJ(DATA, TBL, COL, LTH) $

Purpose: INTOTJ converts an integer value to EBCDIC decimal and places it right-justified, with leading blanks, in a programmer-defined output area. Leading zeros are returned as blanks.

Use: The arguments to this routine are:

DATA – The value to be converted.

TBL and COL – Together represent the address of the first byte of the output area. In some cases, it may be desirable to describe the address completely as TBL, setting COL to zero; in other cases, each may have a nonzero value whose sum represents the desired address.

LTH – The length (size) of the output area in bytes.

INTOTJ is re-entrant.

Restrictions: If the value of DATA is a binary zero, the output is a single zero right-justified in the output area.

LTH, the fourth argument, must be at least 2 and less than or equal to 11. If the LTH field is an invalid length, an asterisk will be placed in the first byte of the output field.

Parameters:

ITEM	DATA	I	32	S	$
ITEM	TBL	I	32	S	$
ITEM	COL	I	32	S	$
ITEM	LTH	I	32	S	$

BAL save area: 24 bytes

Length: 136 decimal bytes

2.1.4.32 Convert Integer Item to EBCDIC with Leading Zeros (INTOTZ) Procedure

Procedure Call Format: INTOTZ(DATA, TBL, COL, LTH) $

Purpose: INTOTZ converts an integer value to EBCDIC decimal and places it right-justified, with leading zeros, in a programmer-defined output area. Leading zeros are returned as zeros.

Use: The arguments to this routine are:

DATA – The value to be converted

TBL and COL – Together represent the address of the first byte of the output area. In some cases, it may be desirable to describe the address completely as TBL, setting COL to zero; in other cases, each may have a nonzero value whose sum represents the desired address.

LTH – The length (size) of the output area in bytes.

INTOTZ is re-entrant.

Restrictions: This routine performs the same functions as INTOTJ except that it returns leading zeros rather than leading blanks.

LTH, the fourth argument, must be at least 2 and less than or equal to 11. If the LTH field is an invalid length, an asterisk will be placed in the first byte of the output field.

Parameters:

ITEM	DATA	I	32	S	$
ITEM	TBL	I	32	S	$
ITEM	COL	I	32	S	$
ITEM	LTH	I	32	S	$

BAL save area: 24 bytes

Length: 120 decimal bytes

2.1.4.33 Left-Justify Hollerith Data (LEFTJ) Function

Function Call Format: LEFTJ(DATA, LTH)

Purpose: LEFTJ left-justifies standard JOVIAL Hollerith data fields which are right-justified with leading zeros.

Use: The arguments to this routine are:

DATA – The right-justified Hollerith item.

LTH – The length in bytes of the item to be left-justified.

The input argument is stored in the rightmost LTH bytes of a work area, and leading zeros are supplied if LTH $<$ 8. The LTH bytes are then examined, one at a time, starting with the leftmost. When a nonzero byte is found, processing is terminated. If a zero is found, the LTH–1 bytes to its right are shifted left one byte, and a trailing blank (X'40') is appended on the right.

The value is returned to the user in an 8-byte output item. While the Hollerith field is now left-justified in its LTH bytes, the 8-byte item will contain 8-LTH leading binary zeros.

LEFTJ is re-entrant.

Restrictions: The input item must be right-justified with leading hexadecimal zeros. The length of the input item cannot be greater than eight bytes.

Parameters:

ITEM	DATA	H	8	$	
ITEM	LTH	I	32	S	$
ITEM	LEFTJ	H	8	$	(O/P)

Length: 40 decimal bytes

2.1.4.34 LOC Function

Purpose: The LOC function allows the JOVIAL user access to the actual address of a JOVIAL-declared data element.

Use: See the JOVIAL User's Manual for a description of the uses and restrictions.

NOTE: This routine is on the library solely to permit the JOVIAL compiler to produce appropriate in-line code.

2.1.4.35 Locate Table Address in Words (LOCW) Function

Function Call Format: LOCW(NAM)

Purpose: LOCW finds the address of a table and makes it available to the JOVIAL programmer.

Use: The argument to this routine is NAM, the name of the table with the desired address.

This routine finds the address of the table specified. LOCW takes the table address divided by 4 as its value.

A common use of the functions is to reference Table DATA using items declared in Table FORM, when FORM is equated to location zero and is declared one word per entry. A programmer might code:

```
BETA = FORMI($LOCW(DATA)$)$
```

LOCW is re-entrant.

Restrictions: To accomplish the intended purpose, the argument of this routine must be a table or array name.

Parameters:

ITEM	NAM	I	32	S	$	
ITEM	LOCW	I	32	S	$	(O/P)

Length: 16 decimal bytes

2.1.4.36 LOGN Function

Purpose: The LOGN function computes the natural logarithm of a floating-point number ($\log^e$ input) greater than zero.

Input: The input is a number greater than zero.

Output: The output is the natural logarithm, an absolute value.

JOVIAL Function Call:

LOGN(xxx)

where:

xxx is the input argument.

BAL Calling Sequence

```
LA      10, Pararea
L       15, = A(ZVLOGN)
BALR    14, 15
```

where:

Pararea is the address of an 8-byte parameter area that contains a normalized floating-point number in the first four bytes and the absolute value of the natural logarithm in the last four bytes, followed by an 18 word working storage area. Pararea must be doubleword aligned.

Range: $0 < \text{input} < +.72370051 \times 10^{76}$
(Negative input numbers are treated as positive; if a zero value is used, a zero answer is returned.)

Accuracy: For natural logarithms less than 1, the absolute error does not exceed 4.8×10^{-7}; for natural logarithms greater than 1, error does not exceed ±4.8 in the seventh significant figure.

2.1.4.37 Loop Control (LTIME) Procedure

Procedure Call Format: LTIME(TIME)

Purpose: LTIME prints the address of the last call to LTIME and the elapsed time in milliseconds since the call.

Use: The argument to this routine is TIME, the number of milliseconds of execution time the program is to be permitted before being terminated by a timer interrupt. If TIME is positive, the program will be terminated via SVC SYSEOJ. If negative, such termination will be via SVC SYSDMP to include a system dump. (Each successive call to LTIME provides a completely new time-to-termination specification.) Intended use of LTIME is to include several calls to the procedure in a program being tested or timed, with the value of each time argument such that normal operation will be permitted but a loop will cause termination.

On the first call, the calling address is noted, the timer is set, and LTIME returns to the caller. On subsequent calls, the time between calls is calculated and the following message is printed:

LAST TIME CALL AT aaaaaa, ELAPSED TIME tttt. ttt SEC

where:

aaaaaa is address of the previous call.

tttt. ttt is the elapsed time in seconds.

(Time devoted to calling the IOPACK print service is not included, but if a related interrupt occurs between calls to LTIME, the indicated elapsed time will be increased by interrupt service time.)

If a timer interrupt occurs, the following messages will be typed and printed and LTIME will exit to the NOSS Monitor:

TIME INTER. AT iiiiii, CALL WAS AT aaaaaa

where:

aaaaaa is the address of the previous call.

iiiiii is the address at which the timer interrupt occurred.

Parameters:

ITEM TIME I 32 S $

Services Used: SYSPRS, SYSTIM, SYSTWR

Length: 352 decimal bytes

2.1.4.38 Compute SIN/COS From ACPs (MCSINE) Procedure

Purpose: This routine is identical to CSINE except for the name of the routine and its subsidiary entry point. It is intended to allow UNTE to load a separate copy of the CSINE procedure into MACH storage; thus a different name must be used.

Use: See CSINE (Subsection 2.1.4.16). The entry point for the table is ZVMSNCOS.

2.1.4.39 Convert Mixed Number to EBCDIC (MIXOTJ) Procedure

Procedure Call Format: MIXOTJ(DATA, TBL, COL, LTH, DP) $

Purpose: MIXOTJ converts a mixed number, "mix," having X significant decimal places after the decimal point, to EBCDIC decimal with decimal point and leading blanks (left of the decimal point) and places it in a programmer-defined output area.

Use: The arguments to this routine are:

DATA – The value to be converted times a power of 10 that will reposition the decimal point X places ($mix*10^X$). This adjustment allows DATA to maintain all necessary significance and yet be defined as an integer.

TBL and COL – Together (added together within MIXOTJ) represent the address of the first byte of the output area.

LTH – The length (size) of the output area in bytes.

DP – X, the number of significant digits after the decimal point, used both for converting DATA and for positioning the EBCDIC decimal point.

A possible use of this routine would be to print a fraction field, as follows:

MIXOTJ(SLICE/PIE*1000,PRNT, 0, 5, 3)$

where:

if SLICE was 1/8 of PIE, the output area would contain 0.125.

Restrictions: An asterisk will be placed in the first byte of the output field if:

LTH is less than 3 or
LTH--DP is less than 2 or
DATA cannot be represented in LTH bytes (+99.9 requires four bytes, --99.9 requires five bytes)

Parameters:

ITEM	DATA	I	32	S	$
ITEM	TBL	I	32	S	$
ITEM	COL	I	32	S	$
ITEM	LTH	I	32	S	$
ITEM	DP	I	32	S	$

BAL save area: 32 bytes

Length: 160 decimal bytes

2.1.4.40 MODULO Function

Purpose: The MODULO function produces an output that is equivalent to input modulo 14400.

Input: The input is a time in units of 1/10 minute. It will generally be of the form TIME1 + TIME2 and must be between – 14400 and 28799 inclusive.

Output: The output is a time in units of 1/10 minute. It will be between 0 and 14399 inclusive.

JOVIAL Function Call:

MODULO(xxx)

where:

xxx is the input argument.

BAL Calling Sequence:

```
LA      10, Pararea
L       15, = A(ZVMODULO)
BALR    14, 15
```

where:

Pararea is the address of a two-word parameter area containing the input in the first word and receiving the output in the second word. Both are 32-bit signed integers.

Range: $-14400 \leq$ input < 28800
$0 \leq$ output < 14400

MODULO is re-entrant.

Length: 40 decimal bytes

2.1.4.41 Move Character (MVC) Procedure

Procedure Call Format: MVC(ADRA, LTH, ADRB)$

Purpose: MVC moves a character field of a specified length from one address to another address.

Use: The arguments to this routine are:

ADRA – The address of the field into which the data are to be moved.

LTH – The length (size) of the field to be moved.

ADRB – The address from which the field is to be copied.

The routine moves a character field byte by byte, from left to right, from one address to another. A typical use of the routine might be moving the day's date from the monitor's date buffer to a print image for a page heading. The user would code:

```
MVC(LOC(HEADER) +95, 8, X(4FC))$
```

where:

Location 4FC is the NOSS monitor's buffer for the day's date. Then the 8-character date would be moved to column 96 of the print image for the page heading. Another common use of the routine would be for zeroing a buffer area as follows:

```
MVC(BUFFER, 1, AZERO)$
MVC(LOC(BUFFER)+1, 249, BUFFER)$
```

where:

AZERO is the address of the constant X(00). Then the 250 bytes beginning at BUFFER would be cleared.

MVC is re-entrant.

Restrictions: The length provided as the second argument must lie in the range $1 \leq$ LTH ≤ 256. If LTH ≤ 0, the routine will return without moving any characters. If LTH > 256, it will be treated modulo 256.

Parameters:

ITEM	ADRA	I	32	S	$
ITEM	LTH	I	32	S	$
ITEM	ADRB	I	32	S	$

BAL save area: 12 bytes

Length: 40 decimal bytes

NOTE: Because code may be generated in-line, null arguments should not be used. For in-line code, an input length of 0 will move 255 bytes.

2.1.4.42 Move Item to Table (MVI) Procedure

Procedure Call Format: MVI(ADRA, LTH, VAL)$

Purpose: MVI moves an immediate character field into an address.

Use: The arguments to this routine are:

ADRA – The address (name of table) of the field into which the data are to be moved.

LTH – The length of the field in bytes.

VAL – The field to be moved.

This routine moves the character field supplied into the address specified.

The typical use of the routine might be moving a character set into a table. The user would code:

```
MVI(LOC(POOL)+10, 6, 6H(REJECT))$
```

where:

The word REJECT is to be moved into the 11th byte of Table POOL. Another common use of the routine would be to move the contents of an item into a table. One might code:

```
MVI(POOL, 7, FPIDN($INDEX$))$
```

where:

The Flight Plan Ident will be moved into temporary storage Table POOL.

MVI is re-entrant.

Restrictions: The length provided as the second argument must lie in the range $1 \leqslant LTH \leqslant 8$. If $LTH \leqslant 0$, the routine will return without moving any characters. If $LTH > 8$, it will be treated modulo 8. The third argument will be used as is if it is unsigned integer/Hollerith. Other data types will be converted to this format by the JOVIAL Compiler.

Parameters:

ITEM	ADRA	I	32	S	$
ITEM	LTH	I	32	S	$
ITEM	VAL	H	8		$

BAL save area: 8 bytes

Services Used: None.

Length: 56 decimal bytes

NOTE: Because code may be generated in-line, null arguments should not be used.

2.1.4.43 Move Unlimited Character (MVM) Procedure

Procedure Call Format: MVM(ADRA, LTH, ADRB)$

Purpose: MVM moves a character field of a specified length (which is unlimited) from one address to another address.

Use: The arguments to this routine are:

ADRA – The address of the field into which the data are to be moved.

LTH – The length (size) of the field to be moved.

ADRB – The address from which the field is to be copied.

This routine moves byte by byte, from left to right, a character field of unlimited length from one address to another.

The typical use of the routine might be moving an input message from a temporary storage location to a working table. The user would code:

MVM(SHCH, BTCNT, LOC(POOL)+(PNTR*4))$

where:

SHCH is the name of working table.

BTCNT is the size of the input message.

POOL is the table containing the work area.

PNTR*4 is the relative location of the work area with Table POOL.

MVM is re-entrant.

Restrictions: There is no upper limit to the value of the second argument LTH, other than that imposed by the size of available memory. If LTH $\leq$ 0, the routine will return without moving any characters.

Parameters:

ITEM	ADRA	I	32	S	$
ITEM	LTH	I	32	S	$
ITEM	ADRB	I	32	S	$

BAL save area: 8 bytes

Length: 64 decimal bytes

2.1.4.44 Move Table to Item (MVT) Function

Function Call Format: MVT(ADRA, LTH)

Purpose: MVT returns a character field of a specified length from an address as the function output value.

Use: The arguments to this routine are:

ADRA – The address of the field from which to move the data.

LTH – The length of the field to be moved.

This routine accesses the specified field by address and length description.

A typical use of the routine might be the setting of a temporary item to the entered departure point in an amendment message. The user would code:

MIDEP = MVT(LOC(POOL)+GAMMA, 4)$

where:

The four-byte departure point name located GAMMA bytes from the start of Table POOL is placed in Item MIDEP.

MVT is re-entrant.

Restrictions: The length provided as the second argument must lie in the range $1 \leqslant LTH \leqslant 8$. If the second argument is specified as less than eight bytes, then the high-order bytes of the item will be set to hexadecimal zeros. If $LTH \leqslant 0$, the output value will be eight bytes of hexadecimal zeros. If $LTH > 8$, it will be treated modulo 8.

Parameters:

ITEM	ADRA	I	32	S	$	
ITEM	LTH	I	32	S	$	
ITEM	MVT	H	8		$	(O/P)

BAL save area: 8 bytes

Length: 64 decimal bytes

2.1.4.45 AND Character (NC) Procedure

Procedure Call Format: NC(ADRA,LTH,ADRB) $

Purpose: NC allows the JOVIAL programmer access to the NC machine operation.

Use: The arguments to this routine are:

ADRA – Destination field address

LTH – Field length ($1 \leqslant LTH \leqslant 256$)

ADRB – Source field address

NOTE: This routine is generated inline by the JOVIAL compiler. It is described here to allow users to find writeups on this and similar routines in a single location. Null arguments are not permitted.

2.1.4.46 AND Item to Table (NI) Procedure

Procedure Call Format: NI (ADRA, LTH, VAL) $

Purpose: NI allows the JOVIAL programmer to utilize the machine AND operation with immediate data.

Use: The arguments to this routine are:

ADRA – Destination field address

LTH – Field length ($1 \leqslant LTH \leqslant 8$)

VAL – the immediate data to be ANDed into the destination field. This value is expected to be Hollerith on unsigned integer format. It will be placed right-justified in an 8-character work area, and the rightmost LTH bytes of this field will be used in the actual operation.

NOTE: Whenever possible, inline code will be generated for this routine. For this reason, null arguments are not permitted. If an actual routine call is used, the code called is re-entrant.

2.1.4.47 OR Character (OC) Procedure

Procedure: Call Format: OC (ADRA, LTH, ADRB) $

Purpose: OC allows the JOVIAL programmer access to the OC machine operation.

Use: The arguments to this routine are:

ADRA – Destination field address

LTH – Field length ($1 \leqslant LTH \leqslant 256$)

ADRB – Source field address

NOTE: This routine is generated inline by the JOVIAL compiler. It is described here to allow users to find writeups on this and similar routines in a single location. Null arguments are not permitted.

2.1.4.48 OR Item to Table (OI) Procedure

Procedure Call Format: OI(ADRA, LTH, VAL) $

Purpose: OI allows the JOVIAL programmer to utilize the machine OR operation with immediate data.

Use: The arguments to this routine are:

ADRA – Destination field address

LTH – Field length ($1 \leqslant LTH \leqslant 8$)

VAL – The immediate data to be ORed into the destination field. This value is expected to be Hollerith or unsigned integer format. It will be placed right-justified in an 8-character work area, and the rightmost LTH bytes of this field will be used in the actual operation.

NOTE: Whenever possible, inline code will be generated for this routine. For this reason, null arguments are not permitted. If an actual routine call is used, the code called is re-entrant.

2.1.4.49 Operational SVC Simulation (OPSVC) Procedure

Procedure Call Format: OPSVC $

Purpose: OPSVC logs all SVCs which are illegal to NOSS on the system output unit. For certain NAS SVCs, the service is simulated. NAS supervisor services and their JOVIAL calls are described for the appropriate model in the NAS Monitor Handbook. This specification is a prerequisite for understanding the use of OPSVC. Services simulated by OPSVC are as follows:

LEASE: An index into the dynamic storage pool (Table POOL) is returned as an output parameter.

RELEAS: The previously LEASEd storage becomes available for further LEASEing.

SEND: The program element number or logical device number is printed on the system output unit, and the message is dumped in hexadecimal. The storage is effectively RELEASed.

ACCEPT: An index for the next queue message in the dynamic storage pool and the message length are returned as output parameters.

POINT: An index into the dynamic storage pool is received as an input parameter, and the next acceptable message is located. The index to this message and its length are returned as output parameters.

SELECT: An index into the dynamic storage pool is received as an input parameter, and the length of the message is returned as an output parameter.

HOLD: The message is returned to the queue for later possible use.

DELETE: The area of the dynamic storage pool that held the message becomes available for later use.

RELAY: The program element number or logical device number is printed on the system output unit, and the message is dumped in hexadecimal. The storage is effectively RELEASED.

ETIME: The current program execution time is returned as an output parameter and is printed on the system output unit.

FINIS, TERMSU, TERMSO: The final program execution time is printed on the system output unit. The job is terminated via the NOSS SYSEOJ routine or the SYSCOM routine for multiple execution passes.

LINK: NAS SVC 104 calls to bufferable programs are supported provided that a suitable DEBUG card is included to force generation of an incore copy of the load map. This card may be coded:

```
10      16
DUMP    MAP
```

in BAL, or punched in object format with a blank in column 19 and a binary zero (12–0–1–8–9) in column 20. This service will require space in Table POOL.

PLOAD: As LINK, but no branch is executed. Instead the load address is returned to the caller.

Use: OPSVC must be called before any operational SVC is executed.

OPSVC users must be running with the operational compool. Specifically, segment ZXPO1 must be loaded. Furthermore, the second halfword of this segment (Table POOL) must be set to the size of POOL (dynamic storage pool) in words. This must be done before the call to OPSVC. The following JOVIAL code will do the job:

```
POWDS ($0$) = NENT(POOL) $
OPSVC $
```

POWDS is the ITEM name in Table POOL. Alternatively, the second halfword of POOL may be set by a data overlay.

If the user does not supply a compool with a deck name or entry point ZXPO1, a block of 4096 words will be provided, with POWDS correctly preset.

Control cards are available to initialize messages in the queue. There are three card types:

/QMESG – Defines a message in the queue.

/QCONT – Contains a message started on a previous /QMESG card.

/QEND – The last card for each initialization of the queue.

Card formats are as follows:

```
col 1.      10      16
                    X'hex coded message'
/QMESG      dddd          or
                    C'BCD coded message'

/QCONT              X'hex coded message'
                          or
                    C'BCD coded message'

/QEND
```

where:

ddd is the size of the storage block needed to hold the message.

X indicates that the message data is in hexadecimal.

C indicates that the message data is in EBCDIC.

/QMESG cards and /QCONT cards are optional; a /QEND card is required. All OPSVC control cards follow the $DATA card.

Control cards are read whenever a call to OPSVC is made. There must be one set of control cards for each call to OPSVC, at least a /QEND card (when there are no queue messages).

On each call to OPSVC, allocation of the storage pool is started fresh. Queue messages and LEASEd storage are not carried over from one OPSVC call to the next.

After the call to OPSVC, calls may be made to any NAS SVC. For all NAS SVCs, the following message will be printed:

```
OPERATIONAL SVC AT ADR xxxxxx,
    CODE   ddd   NNNNNN
```

where:

XXXXXX is the hexadecimal address of the SVC instruction.

ddd is the decimal SVC code.

NNNNNN is the NAS SVC mnemonic.

Additionally, a service may be simulated as described under Purpose. Control is returned to the caller except for SVCs FINIS, TERMSU, and TERMSO.

Restrictions:

a. There must be fewer than 50 /QMESG cards for one OPSVC call (excess ignored).

b. Illegal control card formats will be diagnosed, but will not cause job termination.

c. There must be fewer than 150 LEASEd blocks held at any one time. Violation results in job termination.

d. LEASE requests and queue messages must not exceed POOL size.

e. Validity of input parameters is checked for all simulated services. Invalid parameters are diagnosed, and control is returned to the caller

Services Used: SYSRDS, SYSPIN, SYSPRS, SYSRTA

Length: 7104 decimal bytes

2.1.4.50 Punch Object Card (POBJ) Procedure

Procedure Call Format: POBJ(TBL, ADR)$

Purpose: POBJ provides the checksum and card ID for an object ESD, TXT, or RLD card and punches it out on the auxiliary and/or system output units.

Use: The arguments to this routine are:

TBL – The address (table name) of the control table, the contents of which are defined below.

ADR – The address (table name) of the 80-byte card image.

The items of the control table are:

CT – The card count which is initialized by the user and stepped by POBJ.

PNAM – The four-character EBCDIC name for the card's ID field.

LOGA – The logical unit number of the first output device.

LOGB – The logical unit number of the second output device (zero if only one output device).

An example of the use of this routine is when an assembler puts out an object deck on both the system output unit and the auxiliary tape. The programmer would code:

```
TABLE   TABL  R     1    3     $
  BEGIN
    ITEM    CT      I    32    S    0    0        N    $BEGIN 0 END
    ITEM    NAM     H    4          1    0        N    $BEGIN 4H(OBJP) END
    ITEM    LOGA    I    16    S    2    0        M    $BEGIN 4 END
    ITEM    LOGB    I    16    S    2    1    6   M    $BEGIN 5 END
  END
      .
      .
      .
    POBJ(TABL, CRDIMG)$
```

The card image at CRDIMG would be punched on the auxiliary tape and system output unit with a sequence number of OBJP0000 and with the correct checksum in columns 9 and 10.

The card count is incremented each time POBJ is called, but steps from 9999 to 0000. The next card punched in this example would have a sequence number OBJP0001, unless the user chooses to set the sequence number according to his own rules upon each entry to the routine.

POBJ forms a checksum of columns 1 through 8 and 11 through 72 and places it in columns 9 and 10.

Restrictions: This program does not make provision for any error conditions in the output devices. If an error return is necessary, the user must call SETRET.

The card image area must be 80 bytes long because this routine will alter the 73rd through the 80th bytes.

Parameters:

```
ITEM   TBL     I    32    S    $
ITEM   ADR     I    32    S    $
```

Table Formats:

```
TABLE   TABL  R     1    3     $
  BEGIN
    ITEM    CT      I    32    S    0    0    N    $
    ITEM    PNAM    H    4     S    0    0    N    $
    ITEM    LOGA    I    16    S    2    0    M    $
    ITEM    LOGB    I    16    S    2    16   M    $
  END
```

Services Used: SYSWRS

Length: 240 decimal bytes

2.1.4.51 General Print (PRINT) Procedure

Procedure Call Format: PRINT(UNIT, ADR, LTH = CHECK)$

Purpose: PRINT outputs a print image (or group of blocked print images to a specified output unit (on-line printer tape, or console typewriter) and returns a check value.

Use: The arguments to this routine are:

UNIT – The logical unit number of the output device.

ADR – The address of the carriage control character preceding the image to be printed.

LTH – The length of the image or blocked group of images including carriage control characters and record marks.

CHECK – An output value indicating success or failure of the operations.

Individual print images may be from 1 to 132 characters long. Each is preceded by a carriage control character to control carriage spacing before printing as follows:

+ or &	No space before print
Blank	One space before print (single space)
0	Double space
–	Triple space
1	Restore to new page before printing
2 through 5	Skip to specified carriage tape punch before printing

If desired, several images may be prepared, each with its leading carriage control character, and blocked together with a record mark (X'E0') between pairs of images.

When outputting to tapes other than the system output unit, the format must be variable blocked records with physical record size not more than 65535 bytes, and 9-track, if standard NOSS tape-to-print routines are to be used to print the tape.

CHECK takes the following values, assuming the programmer has appropriately initialized error and end-of-file returns for the tape unit being used (system output print unit and console typewriter always return the value 0):

0 – Success

1 – End of tape

2 – Uncorrectable error

Restrictions: Logical units 3 (system output print unit) and 12 (console typewriter) are systems defined, but any other unit used must have a $UNIT card (problem programs) or be defined via SYSDTF (NOSS processors).

Parameters:

ITEM	UNIT	I	32	S	$
ITEM	ADR	I	32	S	$
ITEM	LTH	I	32	S	$
ITEM	CHECK	I	32	S	$

Services Used: SYSPRS

Length: 32 decimal bytes

2.1.4.52 Punch Text without Zero Cards (PTXNZ) Procedure

Procedure Call Format: PTXNZ(TBL, ADRA, LTH, ADRB)$

Purpose: PTXNZ punches a program that is in storage into one or two output units in TXT card format.

Use: The arguments to this routine are:

TBL -- The address (table name) of the control table.

ADRA -- The address of the first word that is to be punched.

LTH -- The number of bytes (size) to be punched.

ADRB -- The desired assembled address (where the program would be loaded by an absolute loader).

The items of the control table are:

CT -- The card sequence number for the first card to be punched. On return, CT is the card sequence number for the next card to be punched.

PNAM -- The four-character EBCDIC symbol for the deck ID field.

LOGA -- The logical unit number of the first output device.

LOGB -- The logical unit number of the second output device (zero if only one output device).

This routine punches TXT cards that represent the information in sequential locations in storage beginning at ADRA, but assigns TXT card addresses as if the program were located at ADRB. It examines as many locations as LTH specifies. Before each TXT card is outputted, the routine looks for the next sequential nonzero byte. It then moves that byte and the next 55 bytes of storage (assuming LTH has not been exceeded) to the TXT card image. The card image is checksummed, and the card ID and sequence number are inserted. Finally, the card image is outputted on one or two output devices according to LOGA and LOGB.

if a programmer wishes to use this routine to punch out from PROG through PROG+200 and the first 20(14 in hexadecimal) bytes were zero, he would code:

```
TABLE     TABL R   1    3     $   BEGIN
    ITEM  CT       I    32    S   0    0    N   $BEGIN 65 END
    ITEM  PNAM     H     4    1   0         N   $BEGIN 4H(MINE) END
    ITEM  LOGA     I    16    S   2    0    M   $BEGIN 4 END
    ITEM  LOGB     I    16    S   2    16   M   $BEGIN 0 END
                .
                .
                .
                .
PTXNZ(TABL, PROG, 201, X(9000)$
                .
                .
                .
```

The first card image outputted on the system output unit would contain an assembled address of X(9014), a byte count of 56, and a card ID of MINE0065. If there were no more consecutive zero bytes, the last card outputted would have an assembled address of X(90BC), a byte count of 13, and a card ID of MINE0068.

Restrictions:

a. If the sum of ADRA and LTH is greater than the highest addressable storage, the routine will punch TXT cards to the limit of storage and the job will be terminated.

b. This program does not make provision for any error conditions in the output devices. If an error return is necessary, the user must call SETRET.

c. As noted above, PTXNZ does not punch cards for areas containing only zeros.

Parameters:

```
ITEM  TBL     I   32   S   $
ITEM  ADRA    I   32   S   $
ITEM  LTH     I   32   S   $
ITEM  ADRB    I   32   S   $
```

Table Formats:

```
TABLE     TBL      R   1    3    $
  BEGIN
    ITEM  CT       I   32   S    0   0   N   $
    ITEM  PNAM     H    4        1   0   N   $
    ITEM  LOGA     I   16   S    2   0   M   $
    ITEM  LOGB     I   16   S    2   16  M   $
  END
```

Services Used: SYSPUN, SYSWRS, POBJ

Length: 536 decimal bytes

2.1.4.53 *Punch Text (PTXT) Procedure*

Procedure Call Format: PTXT (TBL, ADRA, LTH, ADRB) $

Purpose: PTXT punches a program that is in storage onto one or two output units in TXT card format.

Use: The arguments to this routine are:

TBL – The address (table name) of the control table.

ADRA – The address of the first word to be punched.

LTH – The number of bytes (size) to be punched.

ADRB – The desired assembled address (where the program would be loaded by an absolute loader).

The items of the control table are:

CT – The card sequence number for the first card to be punched. On return, CT is the card sequence number for the next card to be punched.

PNAM – The four-character EBCDIC symbol for the deck ID field.

LOGA – The logical unit number of the first output device.

LOGB – The logical unit number of the second output device (zero if only one output device).

This routine punches TXT cards that represent the information in sequential locations in storage beginning at ADRA, but assigns TXT card addresses as if the program were located at ADRB. It examines as many locations as LTH specifies and punches TXT cards for the complete storage area. Each card image is checksummed, and the card ID and sequence number are inserted. Then the card image is outputted on one or two output devices, according to LOGA and LOGB.

If a programmer wishes to use this routine to punch out from PROG through PROG+200, he would code:

```
TABLE    TABL   R   1    3   $   BEGIN
    ITEM CT     I   32   S   0   0     N   $BEGIN 65 END
    ITEM PNAM   H   4    1       0     N   $BEGIN 4H(MINE) END
    ITEM LOGA   I   16   S   2   0     M   $BEGIN 4 END
    ITEM LOGB   I   16   S   2   16    M   $BEGIN 0 END
           .
           .
           .
PTXT (TABL, PROG, 201, X(9000)$
           .
           .
           .
```

The first card image outputted on the system output unit would contain an assembled address of X(9000), a byte count of 56, and a card ID of MINE0065. The last card output would have an assembled address of X(90A8), a byte count of 33, and a card ID of MINE0068.

Restrictions:

a. If the sum of ADRA and LTH is greater than the highest addressable storage, the routine will punch TXT cards to the limit of storage and the job will be terminated.

b. This program does not make provision for any error conditions in the output devices. If an error return is necessary, the user must call SETRET.

Parameters:

```
ITEM   TBL     I   32    S   $
ITEM   ADRA    I   32    S   $
ITEM   LTH     I   32    S   $
ITEM   ADRB    I   32    S   $
```

Table Formats:

```
TABLE      TBL     R   1    3   $
  BEGIN
    ITEM   CT      I   32   S   0   0   N   $
    ITEM   PNAM    H    4       1   0   N   $
    ITEM   LOGA    I   16   S   2   0   M   $
    ITEM   LOGB    I   16   S   2   16  M   $
  END
```

Services Used: SYSPUN, SYSWRS, POBJ

Length: 512 decimal bytes

2.1.4.54 General Punch (PUNCH) Procedure

Procedure Call Format: PUNCH(UNIT, ADR, LTH–CHECK)

Purpose: PUNCH outputs a punch image to a specified output unit (card punch, console typewriter, tape unit, or printer) and returns a check character

Use: The arguments to this routine are:

UNIT – The logical unit number of the output device.

ADR – The address of the first byte of the card image to be punched.

LTH – The number of characters to be punched.

CHECK – A check character to indicate success or failure of the operation.

This routine outputs a punch image of the specified length from a specified address to the specified logical unit. If the write was successful, CHECK takes the value 0.

If an abnormal condition is encountered during the write operation, and if the programmer has properly initialized error and end-of-file returns for the unit being used, CHECK takes the following values:

1 – An end-of-tape was encountered.

2 – An uncorrectable error was encountered.

An example of use might be:

```
PUNCH(24, PUNIM, 50 = CHECK)$

IF CHECK EQ 0 $ GOTO AOK $

IF CHECK EQ 1 $ GOTO EOT $
```

Upon successful operation, program control would pass to the statement labeled AOK, with PUNIM to PUNIM + 49 written as a card image on logical unit 24.

Restrictions: Logical units 4 and 12 are system-defined basic units (system output and console typewriter), but the characteristics of any non-basic unit used must be specified on a $UNIT card or via SYSDTF (NOSS processors). When using logical unit 5(.AUXIL), use RYTE, not punch.

If the programmer intends to use the check character to indicate abnormal condition, he must initialize by calling SETRET.

Parameters:

ITEM	UNIT	I	32	S	$
ITEM	ADR	I	32	S	$
ITEM	LTH	I	32	S	$
ITEM	CHECK	I	32	S	$

Services Used: SYSPUN

Length: 32 decimal bytes

2.1.4.55 RANDOM Function

Purpose: The RANDOM function generates a pseudo-random floating-point number less than one. The power residue method used is described in the IBM publication Random Number Generation and Testing, Form C20–8011. The following recursive algorithm is used:

$$U_{n+1} = XU_n \pmod{2^b}$$

where:

U_n is a random number

X is a constant multiplier.

b is the word size of the machine.

The initial value of U_n is a 24-bit integer with a 1 in the low- and high-order positions; b is equal to 24 because the output is to be expressed in floating-point; and x is $(4039)_{10}$, satisfying the following relationships: $x = 8t \pm$, where t is any integer and $x = 2^{b/2}$.

The random numbers U_{n+1} are treated as integers during the calculation, but a floating-point characteristic is added to the low-order 24 bits of the result, so that $0 < U_{n+1} < 1$. This floating-point number is placed in the output area. The low-order 24 bits of U_{n+1} replace the previous U_n for use in generating subsequent random numbers.

This function produces approximately 4.2 million pseudo-random numbers before repeating.

Input: The input is any item. (This is a dummy parameter but it cannot be omitted.)

Output: The output is a pseudo-random floating-point number between 0 and 1.

JOVIAL Function Call:

RANDOM(xxx)

where:

xxx is a dummy input argument.

BAL Calling Sequence:

```
LA      10, Pararea              or,
L       15, = A(ZWRANDOM)        L       15, = A(ZWRANDOM)
BALR    14, 15                   CNOP    0, 4
                                 BAL     14, 0(0, 15)
                                 DC      A(Pararea)
```

where:

Pararea is the address of a 2-word parameter area that contains a dummy input parameter in the first word, and will contain the generated pseudo-random number in the last word.

2.1.4.56 Random Integer (RANINT) Function

Function Call Format: RANINT(RANGE)

Purpose: RANINT generates a pseudo-random integer in the range 0 through RANGE–1 for positive values and RANGE through –1 for negative values.

Use: The argument to this routine is RANGE, the limit on size of the generated pseudo-random number.

This routine multiplies a 31-bit fractional seed by a selected multiplier. The fractional portion of the product is stored as a new seed and also is multiplied by RANGE and truncated to generate the output integer RANINT.

A typical use would be to code RANINT(2) to simulate matching coins (1 = match, 0 = no match) or to code RANINT(6)+RANINT(6)+2 to simulated throwing a pair of dice (total 2–12)

Though the seed and multiplier provided are adequate for normal use, they are made accessible for modification in BAL code. The seed is at location ZWRANINT and must be odd. The multiplier is at location ZWRANINT+4, must be odd, and must be carefully chosen if output is to appear random. (Standard multiplier is 5^{13}.)

Parameters:

ITEM	RANGE	I	32	S	$	
ITEM	RANINT	I	32	S	$	(O/P)

Length: 96 decimal bytes

Execution Time: 90 microseconds

2.1.4.57 Read General Input (REED) Procedure

Procedure Call Format: REED(UNIT, ADR, LTH = CHECK) $

Purpose: REED reads no more than the specified number of bytes, from the designated logical unit (card reader, console typewriter, tape unit) into the specified address and returns a check character or word count.

Use: The arguments to this routine are:

UNIT – The logical unit number of the input device.

ADR – The address (table name) where the first byte of information is to be placed.

LTH – The number of bytes of data (size of work area) to be read.

CHECK – The check character for an unsuccessful read or the byte count for a successful read.

This routine reads from the specified logical unit into the specified address. It will never read more bytes than the length argument specifies but will read fewer if the record being read is shorter than the length argument specifies. If the read was successful, CHECK takes the byte count from the read for its value.

If an abnormal condition is encountered during the reading operation, and if the programmer has properly initialized error and end-of-file returns for the input unit being used, CHECK takes the following values:

1 An end-of-file was encountered.

2 – An uncorrectable error was encountered.

A common use of the procedure is to read the first 35 columns of a card image into a buffer area. The programmer might code:

```
REED(2, CRDIM, 35 = CHECK) $
IF CHECK EQ 35$ GOTO AOK. $
IF CHECK EQ 1$ GOTO FINI $
```

Upon a successful read, program control passes to the statement labeled AOK. with the first 35 columns of the card/card image in memory from CRDIM to CRDIM+34.

Restrictions: During the reading of tapes, the length of the logical record being read may be between 3 and 65535 bytes when using buffered I/O operation, 13 to 65535 bytes when using unbuffered operation.

Logical units 2 and 12 are system-defined (system input unit and console typewriter), but any other unit used must have its characteristics specified on a $UNIT card, or by SYSDTF (NOSS Processors).

If the check character is to be used to indicate abnormal condition, the appropriate entry to SETRET must be made before the call to REED. REED can only be used with the Utility monitor. .

Parameters:

ITEM	UNIT	I	32	S	$	
ITEM	ADR	I	32	S	$	
ITEM	LTH	I	32	S	$	
ITEM	CHECK	I	32	S	$	(Reg 1)

Services Used: SYSRDS

Length: 32 decimal bytes

2.1.4.58 Read Tape to Buffer (REEDR) Procedure

Procedure Call Format: REEDR(UNIT = ADR, CHECK)$

Purpose: REEDR reads physical records from tape into buffer areas, specified via $UNIT cards or procedure SYSDTF, and returns the address of the next logical record in the buffer together with a check character or byte count.

Use: The arguments to this routine are:

UNIT – The logical unit number of the input tape.

ADR – The address of the first byte of the logical record in the buffer.

CHECK – The check character for an unsuccessful read or the byte count for a successful read.

This routine reads from the specified tape unit into buffer areas. For this reason, the user has no need of direct control over record length. If the read is successful, ADR takes the address of the record read and CHECK takes the byte length of the record as its value.

If an abnormal condition is encountered during the reading operation, and assuming the programmer has properly initialized error and end-of-file returns for the tape unit being used, CHECK takes the following values:

1 – An end-of-file was encountered.

2 – An uncorrectable error was encountered.

A possible use of the routine is in reading a large volume of long records for which the saving in storage by using the buffer as work area and the saving in time by not moving records from the buffer area is significant.

Restrictions: This routine can only read from buffered tapes. REEDR cannot be used for logical unit 2 or for any other nontape or unbuffered unit.

It should always be assumed that a logical record may start on any byte boundary unless controlled by SYSDTF. The PRS and LRS fields should never be modified in variable blocked records.

If the check character is to be used to indicate abnormal operation, the appropriate entry to SETRET must be made before the call to REEDR. REEDR can only be used with the Utility monitor.

Parameters:

ITEM	UNIT	I	32	S	$
ITEM	ADR	I	32	S	$
ITEM	CHECK	I	32	S	$

Services Used: SYSRDS

Length: 64 decimal bytes

2.1.4.59 REW Procedure

The REW procedure rewinds the selected tape. The procedure call for REW is:

REW(unit) $ (UNIT in Reg 10)

2.1.4.60 Read Console Typewriter (RTYPE) Procedure

Procedure Call Format: RTYPE(ADRA, LTH)$

Purpose: RTYPE reads a message from the console typewriter, providing a flexible communication between the operator and the JOVIAL program.

Use: The arguments to this routine are:

ADRA – The address (name of the table) where the message is to be placed.

LTH – The maximum length message to be accepted. The length will usually equal the number of bytes in Table ADRA (see Restrictions).

The most common use of the procedure will likely be where the programmer is seeking operator aid. For instance, an input card is found to have an illegal format and the programmer wishes to allow the card to be repunched. After communicating the problem, the programmer calls RTYPE, which waits for a typed message. The message is complete when the end-of-message is entered or when the message reaches the specified length. Control is then returned to the calling program, where the reply can be evaluated to determine program action.

To simplify operator action for the most common operator message, CONT (continue), the RTYPE procedure automatically inserts 411 (CONT) into ADRA. This feature allows the operator to enter

end-of-message for normal return and still allows the programmer to perform a single comparison (see Restrictions).

The routine converts all messages to upper case.

Restrictions: The length provided as the second argument of the procedure must lie in the range: $4 \leqslant LTH \leqslant 132$.

This procedure does not clear the input area, but places the 4H(CONT) in the input area before initiating the typewriter read. Consequently, the input message area may not exactly reflect the message typed. For example, if ADRA to ADRA + LTH is blanked before RTYPE is called, some typical results might be:

Typed	Result
1H(E)	4H(EONT)
2H(EX)	4H(EXNT)
3H(EXE)	4H(EXET)
7H(EXECUTE)	7H(EXECUTE)

Similarly, if ADRA is not blanked, the old messages will not be completely overwritten if the new message is shorter.

Parameters:

ITEM	ADRA	I	32	S	$
ITEM	LTH	I	32	S	$

Services Used: SYSTRE

Length: 56 decimal bytes

2.1.4.61 RUN Procedure.

The RUN procedure rewinds and unloads the selected tape. The procedure call for RUN is:

RUN(unit) $ (UNIT in Reg 10)

2.1.4.62 Write General Output (RYTE) Procedure

Procedure Call Format: RYTE(UNIT, ADR, LTH = CHECK) $

Purpose: The RYTE routine writes on the designated logical unit (console typewriter, printer, punch, tape unit) the specified number of bytes from the specified address and returns a check character.

Use: The arguments to this routine are:

UNIT – The logical unit number of the output device.

ADR – The address of the first byte of information to be outputted.

LTH – The number of bytes of data to be outputted.

CHECK – The check character that may be tested to determine the status of the last write operation.

This routine writes on a specified logical output unit the specified number of bytes from the specified address. The check character takes the following values:

0 – Successful write operation.

1 – The operation wrote past the end-of-tape marker.

2 – An uncorrectable error condition was encountered.

This routine would likely be used where the programmer wants to use more than one output device. For example, one wishes to instruct the operator and save the same message on a special output tape. In this case, he would code:

```
RYTE(12, MSG1, 20=) $

RYTE(23,,= CHK) $

IF CHK EQ 0 $ GOTO AOK $

IF CHK EQ 1 $ GOTO EOT $
```

If both writes were successful, the same message would be outputted on two different devices and program control would pass to the statement labeled AOK. (Note that check is not tested on the first write since basic system units never indicate end-of-tape or error to a user, while address and length were omitted from the second call because desired values were identical with the previous call.)

Restrictions: When writing on tape, the record length must be between 16 and 65,535 bytes except for two cases:

1. When fixed length or fixed-length blocked records are being used, the record length is controlled by the $UNIT card and LTH is meaningless; however, record length times the blocking factor must be between 16 and 65535 bytes.

2. When variable-length blocked records are being used, the record length may be between 3 and 65531 bytes.

In all cases, the buffer size must be adequate for the record sizes being used and between 16 and 65535 bytes in length.

Logical units 3, 4 and 12 (system output print unit, system output punch units, and console typewriter) are system-defined, but the characteristics of any other unit must be specified on a $UNIT card or by SYSDTF (NOSS Processors).

If the check character is to be used to indicate abnormal conditions, the appropriate initialization must be accomplished via SETRET before RYTE is called.

Parameters:

ITEM	UNIT	I	32	S	$	
ITEM	ADR	I	32	S	$	
ITEM	LTH	I	32	S	$	
ITEM	CHECK	I	32	S	$	(Reg 1)

Services Used: SYSWRS

Length: 32 decimal bytes

2.1.4.63 NOSS Set I/O Return (SETRET) Procedure

Procedure Call Format: SETRET(UNIT, EOF, ERR) $

Purpose: The SETRET routine provides convenient user control of monitor action taken in case of I/O end-of-file, end-of-tape, or error conditions. If procedure SETRET has not been called, any end-of-file or error condition will result in job termination.

Use: The arguments to this routine are:

UNIT – Logical unit number of device to be affected.

EOF – Address to which control should go in case of end-of-file on input or end-of-tape on output. There are three special cases besides the general case:

0 – End-of-file or end-of-tape is to be considered catastrophic and to result in job termination.

1 – End-of file or end-of-tape is to result in processor termination with an appropriate NOSS monitor comment. (This option is available only to NOSS processors.)

ZWEOFRET – End-of-file or end-of-tape is only to cause CHECK to be set to 1. (If this symbol is used, an EXTRN must by provided as it is an entry to the SETRET routine.)

Any other value, end-of-file or end-of-tape, will result in transfer to this address. The recovery routine placed at this address is expected to take appropriate action. Except for general register 1, which contains the address transferred to, registers have the same value as when IOPACK was called by the BAL code or by a library routine that was used. General register 1 need not be restored and may be used as a base within the recovery routine. All other registers used should be restored, and register 1 should be set to an appropriate value for CHECK (1, 2, or number of bytes), if the programmer decides to return to the original IOPACK exit, which he can do via "B 4(0, 14)." In most cases, however, the program will not return to the original sequence and the programmer will not need to restore registers.

ERR – Address to which control should go in case of uncorrectable error. There are three special cases besides the general case:

0 – Error is to be considered catastrophic and to result in job termination.

1 – Error is to result in processor termination with an appropriate NOSS monitor comment. (This option is available only to NOSS processors.)

ZWERRET – Error is only to cause CHECK to be set to 2. (If this symbol is used, EXTRN must be provided as it is an entry of the SETRET routine.)

Any other value – the same kind of transfer to the recovery routine as for end-of-file.

Restrictions: UNIT may not refer to SYSTEM tape (1) or SYSIN or SYSOUT units (2, 3, 4) or console typewriter (12).

The values of the arguments EOF and ERR must be addresses of appropriate routines. One approach is to reference JOVIAL items in the procedure call and to use direct code to preset the items. Another is to code JOVIAL statement labels as arguments.

Parameters:

```
ITEM   UNIT    I   32   S   $
ITEM   EOF     I   32   S   $
ITEM   ERR     I   32   S   $
```

Services Used: SYSSTR

Length: 48 bytes

2.1.4.64 Sine (SIN) Function

Function Call Format: SIN(THETA)

Purpose: SIN computes the sine of an angle expressed in radians.

Use: The argument to this routine is THETA (the angle in radians). SIN is re-entrant.

Restrictions: Angles less than 2^{-31} radians will be treated as zero; values greater than 2^{32} –1 radians will be truncated.

The absolute error does not exceed 4.8×10^{-7}.

Parameters:

```
ITEM   THETA   F   $
ITEM   SIN     F   $      (O/P)
BAL save area: 44 bytes
```

Doubleword alignment is required.

Length: 400 decimal bytes

2.1.4.65 Super Print (SPRINT) Procedure

Procedure Call Format: SPRINT(CTL, LOC, CHR, LNS) $

Purpose: SPRINT provides a single flexible procedure to output print images together with automatic page heading and numbering.

Printed output generated by most large programs (such as the various NOSS processors) requires page headings. Heading content varies according to the program, but normally includes the date and page number. Column headings are often required and, in many programs, subheadings are required to change at appropriate points in the run. SPRINT is intended to relieve the programmer of the more onerous duties associated with page control and formatting.

Use: The arguments to this routine are:

CTL – The address of the control table, specified by:

1. Any valid address (table name) – SPRINT will expect a user-generated control table to be located at that address. The expected contents of the table are described below.

2. A 0 – SPRINT provides values for the control table as described below.

LOC – The address of the print image, specified by:

1. Any valid address – SPRINT will print from the specified address.

2. A 0 – SPRINT will print from the address specified in Item ADRA of the control table and then clear 133 bytes beginning at that address.

CHR – The number of bytes in the print image (including carriage control character), specified by:

1. A number from 2 to 133 (2–1000 if images are blocked – see below) – SPRINT will print the specified number of bytes.

2. A 0 – SPRINT will not print an image but will initialize to blank characters the 133 bytes beginning at the address specified in Item ADRA of the control table unless that address is 0.

LNS – The number of lines to be occupied by this image, specified by:

1. A positive number SPRINT will assume that the image with its carriage control occupies the indicated number of lines on the page. If this would cause overflow, a heading is printed first to restore to the next page.

2. A negative number – The magnitude will be used in the same way as for a positive number, but overflow will always be assumed.

3. A null – SPRINT will assume the caller set LNS to +1.

The items of the control table are:

PNCOL – The starting column in the first heading image for the page number.

LPP – The number of lines of printed text (not including the heading) per page.

LC – The current line count (SPRINT initializes LC each time it prints a heading to 199–LPP. This provides for normal formats of 50–55 lines and special-purpose formats of up to 200 lines per page).

UNIT The logical I/O unit number.

HC – The number of heading images.

PN – The page number for the next page.

ADRA – The address of the text image when the second argument of the call (LOC) is specified as 0.

BPH – The number of bytes in this heading image including carriage control character.

ADRB – The address of this heading image.

Items BPH and ADRB are repeated the number of times specified in Item HC. When CTL is specified as 0, the control table arguments will take the following values:

PNCOL – 123
LPP – 53
LC – 200
UNIT – 3 (SYSOUT)
HC – 1
PN – 1
ADRA – 0
BPH – 130
ADRB – the address of an internal table containing the image:

ccc	col 20	col 60	col 119
1	PROGRAM OUTPUT	DATE	PAGE

which will be followed by two blank lines.

SPRINT outputs print images and generates automatic page heading and numbering. The first character of every image must be a carriage control character. The carriage control character is used to control spacing before printing the image. SPRINT never spaces after printing. The Carriage Control Characters (CCCs) and their functions are:

CCC	Function
+ or &	No space before printing (overprint this image on the previous image)
Blank	Single-space before printing
0	Double-space before printing
– (minus)	Triple-space before printing
1	Restore page before printing
2–5	Skip to carriage punch 2–5 before printing

All other characters will be interpreted as single space except record mark (X'E0'), which is prohibited from print images, and the letters U, V, W, and X, which are not permitted as carriage control characters. The record mark is used to block several images (see Restrictions) and to present them to SPRINT in a single call. This is accomplished by placing the record mark between the last character of one image and the CCC of the next image. (A record mark should not be placed behind the last image.) Header lines may not be blocked this way.

When the heading is being prepared, the day's date can be found in an 8-byte field at address 0004FC. The CCC for the first line of the heading should be a 1. When SPRINT overflows the line count, it prints the heading before the next line of text. SPRINT does not test the reasonableness of the carriage control character.

If a 1 is used as a CCC in a line of text, the page will be restored but SPRINT will not increment the page count or print the heading. SPRINT can be forced to print the next page heading by using a negative line count (LNS) or by setting the line number in the control table (LC of CTL) to 200. In both cases, the heading will be outputted and the page number incremented before the next image is printed.

When the second argument of SPRINT is 0, the routine will use ADRA of the control table as the address of the print image and blank 133 bytes of that image after printing.

SPRINT must be initialized by call SPRINT (CTL,,0,)$ when the control table is supplied or call SPRINT (0,,0,)$ when SPRINT is to supply the control table.

Several calls to SPRINT are exemplified below. The control table is assumed to be of the format:

```
TABLE     CTL      R     1     WDS    $    BEGIN
  ITEM    PNCOL    I     8     U 0    0    B        $
    BEGIN          100         END
  ITEM    LPP      I     8     U 0    8    B        $
    BEGIN          50          END
  ITEM    LC       I     8     U 0    16   B        $
  ITEM    UNIT     I     8     U 0    24   B        $
    BEGIN          3           END
  ITEM    HC       I     8     U 1    0    B        $
    BEGIN          3           END
  ITEM    PN       I     16    S 1    16   M        $
    BEGIN          1           END
  ITEM    ADRA     U     32    S 2    0    N        $
  STRING  BPH      I     8     U 3    0    B 1 1    $
  STRING  ADRB     I     24    U 3    8    D 1 1    $
                                END
```

where:

WDS = number of header lines + 3.

If the values of ADRA and ADRB have been properly preset, SPRINT (CTL, 0, 120,)$ single-spaces and prints the 119 bytes at address ADRA + 1. This assumes that the CCC at address ADRA is a blank.

SPRINT (CTL, BB, 243, 3)$ prints the three lines beginning at BB. This example assumes that record marks and CCCs have been properly placed within the 243-byte block to be printed.

SPRINT (CTL, BB, 240, 4)$ is used to print two lines if the two CCCs in the print image have been set for double-spacing.

SPRINT (CTL, IMAGE, 120,)$ prints the 119 bytes from address IMAGE + 1.

SPRINT (,,,-1)$ (assuming this call follows the preceding example) prints 119 bytes from address IMAGE + 1 after ejecting a page and printing the headings and subheadings specified in CTL.

PN = 222 $ sets the page number so that the next heading printed will be numbered 222.

LC = 200 $ forces SPRINT to restore the page when it is next called. In the example, there are three heading images. The user could set the heading count HC = 1 $. SPRINT will now print only the first of the three heading images.

Restrictions:

a. An image must be at least two bytes long and must be no longer than 133 bytes, but images may be blocked together (except header lines).

b. Logical I/O unit number 3 is a system-defined unit (the normal job print unit), but any other unit must be specified on a $UNIT card for problem programs or via SYSDTF for system processors.

c. Unit numbers must be the digit 3 for SYSOUT or within the range available to problem programs (20–99) or system processors (6–11).

d. Required $UNIT card format is:

$UNIT TAPE ident, IOPACK, VBL, RL01000, BUF, 9TRACK

or

$UNIT TAPE ident, IOPACK, VAR, RL01000, BUF, 7TRACK, D556, 6BIT, ODD

e. When the second argument (LOC) is zero, ADRA will specify the address of the print image.

f. When SPRINT's internal table is selected by setting CTL to 0, ADRA cannot be modified in JOVIAL, but can be modified in direct code as follows:

```
L       2, = A(ZVSPRCTL)

EXTRN   ZVSPRCTL

L       1, = A(A1BCD3)

ST      1, 80(0, 2)
```

so that ADRA will contain the address of the user-defined table BCD3, which must be 34 words long.

Parameters:

```
ITEM   CTL     I   32    S   $
ITEM   LOC     I   32    S   $
ITEM   CHR     I   32    S   $
ITEM   LNS     I   32    S   $
```

Table Formats:

```
TABLE CTL  R          1    WDS   $
  BEGIN
    ITEM   PNCOL   I   8    U   0   0   B        $
    ITEM   LPP     I   8    U   0   8   B        $
```

```
ITEM     LC     I   8     U   0   16   B              $
ITEM     UNIT   I   8     U   0   24   B              $
ITEM     HC     I   8     U   1   0    B              $
ITEM     PN     I   16    S   1   16   M              $
ITEM     ADRA   I   32    S   2   0    N              $
STRING   BPH    I   8     U   3   0    B   1   1      $
STRING   ADRB   I   24    U   3   8    D   1   1      $
END
```

where:

WDS = number of header lines wanted + 3.

Services Used: SYSPRS

Length: 600 decimal bytes

2.1.4.66 Square Root (SQRT) Function

Function Call Format: SQRT(VAL)

Purpose: SQRT computes the square root of a positive number.

Use: The argument to this routine is VAL, the value to be processed. SQRT is re-entrant.

Restrictions: If a negative argument is supplied, SQRT will return the square root of its magnitude.

Parameters:

```
ITEM   VAL     F   $
ITEM   SQRT    F   $      (O/P)
BAL save area: 32 bytes
```

Doubleword alignment is required.

Length: 312 decimal bytes

2.1.4.67 STF Procedure

The STF procedure spaces forward one file on the selected tape. The tape will be positioned immediately after the tapemark that stopped the operation:

STF(unit) $ (UNIT in Reg 10)

2.1.4.68 STR Procedure

The STR procedure spaces forward one record on the selected tape and checks for an end-of-file condition. This operation is valid for an unbuffered unit only. An uncorrectable error causes termination of the job. The procedure call for STR is:

STR(unit = check) $ (UNIT in Reg 10, CHECK in Reg 1)

where:

Check is a 32-bit unsigned integer item that will be set to 1 if an end-of-file condition is sensed during the operation and 0 if no end-of-file is sensed.

2.1.4.69 System Communications (SYSCOM) Procedure

Procedure Call Format: SYSCOM(FLAGS, NAME, VAL) $

Purpose: SYSCOM provides a communications interface between NOSS processors and the NOSS Monitor.

Use: The arguments to this routine are:

FLAGS – A set of indicators specifying the type of data or request. In some cases, the flag is the complete message; in other cases, the NAME or VAL arguments provide additional details. Legal values for FLAGS are:

X(0001) – Read the record specified in NAME, and transfer to it.

X(0002) – Read the record specified in NAME, and return control to this processor.

X(0005) – Read the record specified in NAME, correct it with patches read from SYSIN, and transfer to it.

X(0006) – Read the record specified in NAME, correct it with patches read from SYSIN, and return control to this processor.

X(0008) – Modify SYSDUMP limits (i.e., start and end dump limits).

X(0100) – Success – final exit from this processor. (With this flag, SYSCOM becomes synonomous with JOVIAL "STOP$" or BAL "SVC SYSEOJ".)

X(0200) – Failure – auxiliary tape output is not usable for problem program execution, system tape generation, etc.

X(0400) – Failure because of a serious failure, appropriately described in system output, this job should be terminated.

X(0800) – Failure – because of a serious error, not adequately described in system output, this job should be terminated with a normal (via SYSOUT) storage dump for analysis. (With this flag, SYSCOM becomes synonomous with BAL "SVC SYSDMP".)

X(1000) – Catastrophic failure – the error detected indicates probable machine or system failure, so NOSS operation should be terminated, but a storage dump would not be meaningful.

X(2000) – Catastrophic failure – the error detected indicates probable machine or system failure so NOSS operation should be terminated with an on-line storage dump for analysis. (The on-line storage dump routine does not use any NOSS Monitor services and, therefore, usually can operate even when IOPACK, MONIO, etc., cannot operate successfully)

NAME – (A) The six-character name of the record desired when using SYSCOM to read from the system tape. (Names of less than six characters must have trailing blanks.) (B) New start address for SYSDUMP. (C) Otherwise, NAME is not required.

VAL – New end address for SYSDUMP.

When flag code 5 or 6 is used to read patch cards, only TXT and REP cards can be read and the load routine does no relocation. At the end of the patch deck, a card with *BEGIN is in columns 1–6 to tell the patch routine to restart normal operation.

When failure is indicated through SYSCOM, a code appropriate to the actual error is selected.

Restrictions:

a. Only NOSS processors are permitted to use SYSCOM (including MOD = SYST).

b. The ability to read patch cards is strictly a debugging service and will not be available in normal operation.

c. As a library routine, SYSCOM will be at a different location in storage from the calling sequence. When FLAG option 2 or 6 is used and control is expected to be returned, the new record should not overlay the calling sequence or the SYSCOM routine.

In some cases, it may be appropriate to use direct code and communicate directly with the NOSS Monitor rather than via the library routine. To do this, a one, three, or four-word argument area (according to number of arguments actually used) is set up and the following sequence coded:

```
         CNOP      2, 4
         SVC       SYSCOM
         DC        A(ARG)

ARG      DS    F   flags
         CNOP  2, 4
         DS    CL6 name or start location (if needed)
         DS    F   val or end location (if needed)
```

or

Alternate Form

```
ARG      DS    F   flags
         DS    F   unused
         DS    F   new start
                   dump loc.
         DS    F   new end
                   dump loc.
```

Parameters:

ITEM	FLAGS	I	32	U	$	
ITEM	NAME	H	6		$	"NAME or START Dump location"
ITEM	SLOC	I	32	S	$	"START Dump location'
ITEM	VAL	I	32	S	$	"END Dump location"

Services Used: SYSCOM (a supervisor call routine of the same name).

Length: 16 decimal bytes

2.1.4.70 Set Up I/O Record Format (SYSDTF) Procedure

Procedure Call Format: SYSDTF(UNIT, MODE, BADR, BLTH, BFAC)$

Purpose: SYSDTF (the JOVIAL library routine) rearranges arguments presented and uses SYSDTF (the supervisor call) to ask the NOSS Monitor to make actual record format entries in the LISTIO.

NOTE: A library routine is provided because SVC SYSDTF requires a parameter list format that cannot be used in JOVIAL code.

Use: The arguments to this routine are:

UNIT – The logical unit number of the device

MODE – The type of record format as follows:

0 – Fixed-length unblocked records (FIX)

1 – Fixed-length blocked records (FXB)

2 - Variable-length records (VAR)

3 – Variable-length blocked records (VBL)

BADR – The address of the buffer area (0 if unbuffered)

BLTH – The size of buffer in bytes (0 if unbuffered)

BFAC – The number of logical records per physical record in FXB mode, unused in other modes.

The intended use of this routine is by system processors (programs residing on the NOSS system tape) to specify file formats which would be specified by $UNIT cards for problem programs.

Restrictions: The unit must be idle. (SYSDTF could follow a rewind or unbuffered read but could not immediately follow a buffered read.)

BADR and BLTH must be zero for unbuffered operation. The two buffers will occupy a space twice the size of BLTH, starting at address BADR.

For FXB mode, BLTH must equal BFAC times logical record length.

Parameters:

ITEM	UNIT	I	32	S	$
ITEM	MODE	I	32	S	$
ITEM	BADR	I	32	S	$
ITEM	BLTH	I	32	S	$
ITEM	BFAC	I	32	S	$

Services Used: SYSDTF (a supervisor call routine of the same name).

Length: 192 decimal bytes

2.1.4.71 TIMDIF Function

Purpose: The TIMDIF function accepts an input time and produces its equivalent with absolute value less than 7200.

Input: The input is a time in units of 1/10 minute. It will generally be of the form TIME1–TIME2 and must have absolute value less than 14400.

Output: The output is a time in units of 1/10 minute with absolute value less than 7200.

JOVIAL Function Call:

TIMDIF(xxx)

where:

xxx is the input argument.

BAL Calling Sequence:

```
L       10, = A(Pararea)
L       15, = A(ZVTIMDIF)
BALR    14, 15
```

where:

Pararea is the address of a two-word parameter area containing the input in the first word and receiving the output in the second word. Both are 32-bit signed integers.

Range: $-14400 \leqslant \text{input} \leqslant 14400$ $-7200 \leqslant \text{output} \leqslant 7200$

TIMDIF is re-entrant.

2.1.4.72 Execute TRT plus Move and TR (TMTR) Procedure

Procedure Call Format: TMTR(ADRA, LTH, ADRB, ADRC, ADRD = CK)

Purpose: TMTR executes MVC, TRT, and TR instructions in sequence.

Use: The arguments to this routine are:

ADRA – The address of the field of data to be moved to, tested, and translated.

LTH – The length of the field of data to be moved, tested, and translated.

ADRB – The address of the translate table to be used for the translate.

ADRC – The address of the translate table to be used for the Translate and Test.

ADRD – The address of the field of data to be used.

CK – The address of the first nonzero byte in the data field (zero if no such byte is found).

This routine uses the 9020 MVC, TRT, and TR instructions to move a data field, translate and test the data field, and translate the data field.

Restrictions: The length provided as the second argument must lie in the range $1 \leqslant LTH \leqslant 256$.

Parameters:

ITEM	ADRA	I	32	S	$
ITEM	LTH	I	32	S	$
ITEM	ADRB	I	32	S	$
ITEM	ADRC	I	32	S	$
ITEM	ADRD	I	32	S	$
ITEM	CK	I	32	S	$

BAL save area: 6 words

Length: 56 decimal bytes

2.1.4.73 Translate (TR) Procedure

Procedure Call Format: TR(ADRA, LTH, ADRB)$

Purpose: TR translates character data from one code to another according to a user-supplied translate table.

Use: The arguments to this routine are:

ADRA – The address of the field of data to be translated.

LTH – The length of the field to be processed.

ADRB – The address of the translate table to be used.

This routine translates the data field using the 9020 TR instruction.

Restrictions: The length provided as the second argument must lie in the range $1 \leqslant LTH \leqslant 256$. Omitted arguments are not permitted because in-line code may be generated.

Parameters:

ITEM	ADRA	I	32	S	$
ITEM	LTH	I	32	S	$
ITEM	ADRB	I	32	S	$

BAL save area: 8 bytes

Length: 32 decimal bytes

2.1.4.74 *Translate and Test (TRT) Procedure*

Procedure Call Format: TRT(ADRA, LTH, ADRC = CK)

Purpose: TRT tests character data by executing a TRT instruction and returns the address of the first byte whose translated value is nonzero.

Use: The arguments to this routine are:

ADRA – The address of the field of data to be tested.

LTH – The length of the field to be processed.

ADRC – The address of the translate table to be used.

CK – The address of the first nonzero byte in the data field (0 if no such byte is found).

Restrictions: The length provided as the second argument must lie in the range $1 \leqslant LTH \leqslant 256$.

Parameters:

ITEM	ADRA	I	32	S	$
ITEM	LTH	I	32	S	$
ITEM	ADRC	I	32	S	$
ITEM	CK	I	32	S	$

BAL save area: 4 words

Length: 32 decimal bytes

2.1.4.75 *Execute Test-and-Set (TSET) Function*

Function Call Format: TSET(ADDR)

Purpose: TSET executes a Test-and-Set (TS) instruction and returns a condition code.

Use: The input argument is the address of the byte to be tested. Returned condition code is 0 or 1, set as the hardware condition code from the TS instruction. TSET is re-entrant.

Parameters:

ITEM	ADDR	I	32	S	$	
ITEM	TSET	I	32	S	$	(O/P)

2.1.4.76 *Execute TRT and TR (TSTR) Procedure*

Procedure Call Format: TSTR(ADRA, LTH, ADRB, ADRC = CK)

Purpose: TSTR executes the TRT and TR instructions in sequence.

Use: The arguments to this routine are:

ADRA – The address of the field of data to be tested and translated.

LTH – The length of the field to be translated and tested.

ADRB – The address of the translate table to be used for the translate.

ADRC – The address of the translate table to be used for the translate and test.

CK – The address of the first nonzero byte in the data field (zero if no such byte is found).

This routine uses the 9020 TRT and TR instructions to translate and test the data field and to translate the data field.

Restrictions: The length provided as the second argument must lie in the range $1 \leqslant LTH \leqslant 256$.

Parameters:

ITEM	ADRA	I	32	S	$
ITEM	LTH	I	32	S	$
ITEM	ADRB	I	32	S	$
ITEM	ADRC	I	32	S	$
ITEM	CK	I	32	S	$

BAL save area: 5 words

Length: 48 decimal bytes

2.1.4.77 TYPE Procedure

The TYPE procedure types a message of a specified number of characters onto the console typewriter. The procedure call for TYPE is:

TYPE(Table, First, Number) S

where:

Table is the name of the table containing the message.

First is an unsigned integer item or constant that specifies the character position of the first character to be typed.

Number is an unsigned integer item or constant that specifies the number of characters to be typed.

2.1.4.78 TYPEW Procedure

The TYPEW procedure types a message of a specified number of characters onto the console typewriter and waits until the operator causes program control to resume. The procedure call for TYPEW is:

TYPEW(Table, First, Number) S

where:

Table is the name of the table containing the message.

First is an unsigned integer item or constant that specifies the character position of the first character to be typed.

Number is an unsigned integer item or constant that specifies the number of characters to be typed.

2.1.4.79 Read Variable Blocked Input Tape and Move Logical Records to User Area (VREED) Routine

Procedure Call Format: VREED(TBL, ADR, LTHL = CHECK)$

Purpose: VREED reads variable blocked input tapes and returns the next logical record to the user work area together with its associated byte count or an error indicator assuming error and end-of-file returns have been properly initialized (i.e., SETRET).

Use: The arguments to this routine are:

TBL – Address of a control table for VREED (described below).

ADR – Address of user work area where logical record is to be returned.

LTHL – Maximum number of bytes of data to be read (i.e., moved to user work area pointed to by ADR).

CHECK – Logical record length of data at ADR for a successful read or an error indicator for an unsuccessful read.

The items in the control table pointed to by TBL are:

UNIT – Logical unit number of input tape.

ADRB – Current working address in the I/O buffer, referenced only by VREED.

LTH – Number of bytes remaining in the physical record currently being deblocked. It MUST be set to 0 (zero) by the calling program before its first call to VREED or after tape repositioning (such as STF). It is normally updated by VREED after initialization.

VREED returns a logical record from the specified logical unit into the specified address. It will never return a logical record longer than the length specified by LTHL, but will return any logical record from three bytes in length up to a record LTHL bytes in length. CHECK contains the byte length of the logical record returned.

If an abnormal condition is encountered during execution of VREED, and if SETRET has been properly initialized for error and end-of-file returns for the logical input being used, CHECK takes on the following values:

0 – Illogical PRS (physical record size) or LRS (logical record size) was detected by VREED logic. When CHECK = 0, ADRB contains the address of the physical record in the I/O buffer and LTH is set to 0 (zero).

1 – An end-of-file was encountered. When CHECK = 1, LTH = 0 and ADRB contains a pointer to LRS for the last successful call to VREED.

2 – An uncorrectable I/O error was encountered. When CHECK = 2, LTH = 0, and ADRB contains a pointer to LRS for the last successful cal to VREED.

Example: An example of VREED usage would be to read the first logical record from logical unit 24:

```
UNIT = 24$
LTH = 0$
VREED (TABL, RDAREA, 1100 = CHECK)$
IF CHECK EQ 0$
GOTO FMTERR$          "BAD PRS/LRS"
IF CHECK EQ 1$
GOTO EOF$             "EOF ENCOUNTERED"
IF CHECK EQ 2$
GOTO IOERR$           "UNCORRECTABLE I/O ERROR"
PROCESS               "CHECK = LRS"
```

Parameters:

```
ITEM   TBL     I   32    S   $   "LOC(TABL)"
ITEM   ADR     I   32    S   $   "LOC(READ AREA)"
ITEM   LTHL    I   32    S   $   "MAX LRS"
ITEM   CHECK   I   32    S   $   "BYTE COUNT OR ERROR INDICATOR"
```

Control Table Format:

```
TABLE     TABL    R   1    3   S
  BEGIN
    ITEM  UNIT    I   32   S   0   0   $   "LOGICAL UNIT"
    ITEM  ADRB    I   32   S   1   0   $   "POINTER IN I/O BUFFER"
    ITEM  LTH     I   32   S   2   0   $   "RESIDUE LEFT IN I/O BUFFER"
  END
```

Restrictions: Note the restrictions which follow.

Input tape records must be variable blocked format.

Mode of reading specified to the monitor via $UNIT card or SYSDTF call must be variable, unblocked, buffered. The user should be informed that declaring the tape as variable-unblocked is required since VREED reads the tape as variable-unblocked and then de-blocks the physical record itself to obtain each logical record. In this way the user may obtain error indicators and diagnostic pointers when error conditions are detected rather than a system abort.

VREED is serially reusable – not re-entrant.

VREED is to be used only with the NOSS Monitor.

Services Used: SYSRDS

Length: 240 decimal bytes

2.1.4.80 *Read Variable Blocked Input Tape and Return Address of Logical Record in Buffer (VREEDR) Routine*

Procedure Call Format: VREEDR(TBL = ADR, CHECK)$

Purpose: VREEDR reads variable blocked input tapes and returns the address of the next logical record (in I/O Buffer) together with its associated byte count or an error indicator assuming error and end-of-file returns have been properly initialized (i.e., SETRET).

Use: The arguments to this routine are:

TBL -- Address of a control table for VREEDR (described below).

ADR -- Address of logical record in I/O buffer.

CHECK -- Logical record length of data at ADR for a successful read or an error indicator for an unsuccessful read.

The items in the control table pointed to by TBL are:

UNIT -- Logical unit number of input tape.

ADRB -- Current working address in the I/O buffer, referenced only by VREEDR.

LTH -- Number of bytes remaining in the physical record currently being deblocked. It MUST be set to 0 by the calling program before its first call to VREEDR or after tape positioning (such as STF). It is normally updated by VREEDR after initialization.

If an abnormal condition is encountered during execution of VREEDR, and if SETRET has been properly initialized for error and end-of-file returns for the logical input being used, CHECK takes on the following values:

0 -- Illogical PRS (physical record size) or LRS (logical record size) was detected by VREEDR logic. When CHECK = 0, ADRB contains the address of the physical record in the I/O buffer and LTH is set to 0 (zero).

1 -- An end-of-file was encountered. When CHECK = 1, and ADRB contains pointer to LRS for last successful call to VREEDR.

2 -- An uncorrectable I/O error was encountered. When CHECK = 2, LTH = 0, and ADRB contains pointer to LRS for last successful call to VREEDR.

Example: An example of VREEDR usage is to read the first logical record from logical unit 24:

```
UNIT = 24$
LTH = 0$
VREEDR (TABL = ADR, CHECK)$
IF CHECK EQ 0 $
GOTO FMTERR $          "BAD PRS/LRS"
IF CHECK EQ 1 $
GOTO EOF $             "EOF ENCOUNTERED"
IF CHECK EQ 2
GOTO IOERR $           "UNCORRECTABLE I/O ERROR"
PROCESS                "CHECK = LRS"
```

Parameters:

ITEM	TABL	I	32	S	$	"LOC(TABL)"
ITEM	ADR	I	32	S	$	"A(LOGICAL RECORD)"
ITEM	CHECK	I	32	S	$	"BYTE COUNT OR ERROR IND."

Control Table Format:

```
TABLE     TABL   R   1    3   $
  BEGIN
    ITEM  UNIT   I   32   S   0   0   $   "LOGICAL UNIT"
    ITEM  ADRB   I   32   S   1   0   $   "POINTER IN I/O BUFFER"
    ITEM  LTH    I   32   S   2   0   $   "RESIDUE LEFT IN I/O BUFFER"
  END
```

Restrictions: Note the restrictions which follow.

Input tape records must be variable blocked format.

Mode of reading specified to the monitor via $UNIT card or SYSDTF call must be variable unblocked, buffered. The user should be informed that declaring the tape as variable-unblocked is required since VREEDR reads the tape as variable-unblocked and then de-blocks the physical record itself to obtain each logical record. In this way the user may obtain error indicators and diagnostic pointers when error conditions are detected rather than a system abort.

VREEDR is serially reusable – not re-entrant.

VREEDR is to be used with the NOSS Monitor.

Services Used: SYSRDS

Length: 176 decimal bytes

2.1.4.81 WEF Procedure

The WEF procedure writes a tapemark on the selected tape. If buffers are used, they are emptied before the tapemark is written. The procedure call for WEF is:

WEF(unit) $ **(UNIT in Reg 10)**

2.1.4.82 WRE Procedure

The WRE procedure writes a tapemark and rewinds the selected tape. If buffers are used, they are emptied before the tapemark is written. The procedure call for WRE is:

WRE(unit) $ **(UNIT in Reg 10)**

2.1.4.83 WRU Procedure

The WRU procedure writes a tapemark and rewinds and unloads the selected tape. If buffers are being used, they are emptied before the tapemark is written. The procedure call for WRU is:

WRU(unit) $ **(UNIT in Reg 10)**

2.1.4.84 Exclusive OR Character (XC) Procedure

Procedure Call Format: XC(ADRA, LTH, ADRB) $

Purpose: XC allows the JOVIAL programmer access to the XC machine operation.

Use: The arguments to this routine are:

ADRA – Destination field address

LTH – Field length ($1 \leqslant LTH \leqslant 256$)

ADRB – Source field address

NOTE: This routine is generated inline by the JOVIAL compiler. It is described here to allow users to find writeups on this and similar routines in a single location. Null arguments are not permitted.

2.1.4.85 Exclusive OR Item to Table (XI) Procedure

Procedure Call Format: XI(ADRA, LTH, VAL) $

Purpose: XI allows the JOVIAL programmer to utilize the machine exclusive OR operation with immediate data.

Use: The arguments to this routine are:

ADRA – Destination field address

LTH – Field length ($1 \leqslant LTH \leqslant 8$)

VAL – The immediate data to be exclusive ORed into the destination field. This value is expected to be Hollerith or unsigned integer format. It will be placed right-justified in an 8-character work area, and the rightmost LTH bytes of this field will be used in the actual operation.

NOTE: Whenever possible, inline code will be generated for this routine. For this reason, null arguments are not permitted. If an actual routine call is used, the code called is re-entrant.

2.2 COMPILER–CALLED ROUTINES

The JOVIAL exponentiation and internal format conversion routines are called only by the compiler; they are not available to the JOVIAL programmer. However, a programmer writing a BAL source program may use these routines by including the given calling sequence in his program; the result is stored in floating-point register 6, except for ZVFIX, which stores the result in general register 1.

2.2.1 Floating-Point Base to Floating-Point Exponent (EXPFF) Routine

Function Call Format: (BAL)

```
LE      6, BASE
LE      0, EXPON
LA      10, ZZ400001
L       15, = A(ZVEXPFF)
BALR    14, 15
```

Use: The arguments to this routine are:

BASE – The floating-point base.

EXPON – The floating-point exponent.

ZVEXPFF is re-entrant.

Restrictions: None

Parameters:

```
ZZ400001    DS    18F    (doubleword alignment is required)
```

Services Used: None

Length: 760 decimal bytes

2.2.2 Floating-Point Base to Integral Exponent (EXPFI) Routine

Function Call Format: (BAL)

```
LE      6, BASE
L       1, EXPON
LA      10, ZZ400001
L       15, = A(ZVEXPFI)
BALR    14, 15
```

Use: The arguments to this routine are:

BASE – The floating-point base.

EXPON – The integral exponent.

ZVEXPFI is re-entrant.

Restrictions: None

Parameters:

```
ZZ400001    DS    9F    (doubleword alignment is required)
```

Services Used: None

Length: 248 decimal bytes

2.2.3 Floating to Fixed-Conversion (FIX) Routine

Function Call Format: BAL

```
LA      10, TMP
LE      6, NUMBER
L       1, SCALE
L       15, = A(ZVFIX)
BALR    14, 15
```

Use: The arguments to this routine are:

TMP – Temporary storage supplied by calling program.

NUMBER – The floating-point number to be converted.

SCALE – The desired scaling factor for the conversion.

ZVFIX is re-entrant.

Restrictions: ZVFIX rounds the low order bit of the output value. Output is in general register 1. Results are unpredictable for ABS (NUMB) greater than $2^{(31-SCALE)}-1$.

Parameters:

```
TMP    DS    2F
```

Services Used: None

Length: 80 decimal bytes

2.2.4 Fixed A-Type to Floating Conversion (FLTA) Routine

Function Call Format: BAL

```
L       0, SCALE
LA      10, TMP
L       1, NUMBER
L       15, = A(ZVFLTA)
BALR    14, 15
```

Use: The arguments to this routine are:

TMP – Temporary storage supplied by calling program.

NUMBER – Fixed-point number with internal binary point.

SCALE – The desired scaling factor for the conversion.

ZVFLTA is re-entrant.

Restrictions: Output is a single-precision floating-point number returned in floating point register 6.

Parameters:

```
TMP   DS    4F    (doubleword alignment is required)
```

Services Used: None

Length: 96 decimal bytes

2.2.5 Fixed Integer to Floating Conversion (FLTI) Routine

Function Call Format: BAL

```
LA      10, TMP
L       1, NUMBER
L       15, = A(ZVFLTI)
BALR    14, 15
```

Use: The arguments to this routine are:

TMP – Temporary storage supplied by calling program.

NUMBER – A binary number with a binary point to the immediate right.

ZVFLTI is re-entrant.

Restrictions: Output is a single-precision floating-point number returned in floating point register 6.

Parameters:

```
TMP   DS    2F    (doubleword alignment is required).
```

Services Used: None

Length: 68 decimal bytes